Tiva Based Daughter Board for Firebird V Hardware And Software Manual.

eRTS Lab IIT Bombay

July 4, 2017

1 Credits

Version 1.0
July 4, 2017

Documentation Author(Alphabetical Order):

1. Ayush Gaurav, Intern eYSIP 2017
2. Nagesh K, Intern eYSIP 2017

Credits(Alphabetical Order):

1. Prof Kavi Arya, CSE IIT Bombay
2. Nex Robotics Pvt. Ltd.
3. Piyush Manavar, Team e-Yantra
4. Saurav Shandilya, Team e-Yantra

2 Notice

The contents of this manual are subject to change without notice. All efforts have been made to ensure the accuracy of contents in this manual. However, should any errors be detected, e-yantra welcomes your corrections. You can send us your queries / suggestions at Contact Us

Table Of Content

3 Introduction

Tiva Daughter board for Fire Bird V will help you gain exposure to the world of robotics and embedded systems with ARM Cortex M4. The board is designed with Open Source Philosophy in software and hardware design ,you will be able to create and contribute to complex applications that run on this platform, helping you acquire expertise as you spend more time with them.

3.1 Safety precautions:

- Robot's electronics is static sensitive. Use robot in static free environment.
- Read the assembling and operating instructions before working with the robot.
- If robot's battery low buzzer starts beeping, immediately charge the batteries.
- To prevent fire hazard, do not expose the equipment to rain or moisture.
- Refrain from dismantling the unit or any of its accessories once robot is assembled.
- Charge the NiMH battery only with the charger provided on the robot.
- Never allow NiMH battery to deep discharge.
- Mount all the components with correct polarity.
- Keep wheels away from long hair or fur.
- Keep the robot away from the wet areas. Contact with water will damage the robot.
- To avoid risk of fall, keep your robot in a stable position.
- Do not attach any connectors while robot is powered ON.
- Never leave the robot powered ON when it is not in use.
- Disconnect the battery charger after charging the robot.

3.2 Inappropriate Operation:

Inappropriate operation can damage your robot. Inappropriate operation includes, but is not limited to:

- Dropping the robot, running it off an edge, or otherwise operating it in irresponsible manner.
- Interfacing new hardware without considering compatibility.
- Overloading the robot above its payload capacity.

- Exposing the robot to wet environments.
- Continuing to run the robot after hair, yarn, string, or any other item is entangled in the robot's axles or wheels.
- All other forms of inappropriate operations.
- Using robot in areas prone to static electricity.
- Read carefully paragraphs marked with caution symbol.

4 Tiva Based Daughter Board

There are two daughter boards one with the launchpad and other one with the Arm Cortex M4 based uC. Almost all the specification are same unless mentioned otherwise.

4.1 Technical Specification

Microcontroller:
TM4C123gh6pm (ARM architecture based Microcontroller)
To know more about the microcontroller please refer to datasheet.

Sensors:
Three white line sensors (extendable to 7)
Five Sharp GP2Y0A02YK IR range sensor (One in default configuration)
Eight analog IR proximity sensors
Two position encoders

Indicators:
2 x 16 Characters LCD
Buzzer

Communication:
USB Communication
Wireless ZigBee Communication (2.4GHZ) (if XBee wireless module is installed)
Bluetooth communication(Can be interfaced on external UART0 available on the board)
Simplex infrared communication (From infrared remote to robot)
I2C Communication

Battery Life:
2 Hours, while motors are operational at 75% of time

Locomotion:
Two DC geared motors in differential drive configuration and caster wheel at front as support
Top Speed: 24 cm / second
Wheel Diameter: 51mm
Position encoder: 30 pulses per revolution
Position encoder resolution: 5.44 mm

5 Hardware Manual:

5.1 Voltage Regulation on the Daughter Board

The voltage source available on the Firebird is 9.6V. But the TIVA platform works on 3.3V and the servos can operate upto 6V. So there must be 3 different voltage levels on the board. The uC based board has 2 voltage regulators and the plug and play board has 1 voltage regulator. In the uC based board the 9.6 volts is 3.3V to power the microcontroller. In the plug and play board the there is an inbuilt voltage regulator, so it is directly connected connected to 5v, 300mA source. The servo in both the boards has a separate 5V regulator.

5.1.1 Powering Micro-controller

The boards have different powering circuits.In the plug and play board is connected to 5V source on Pin 10. In the uC based board the 9.6V source available on Pin 29 is reduced to 3.3V. Refer to the schematic below for further details.

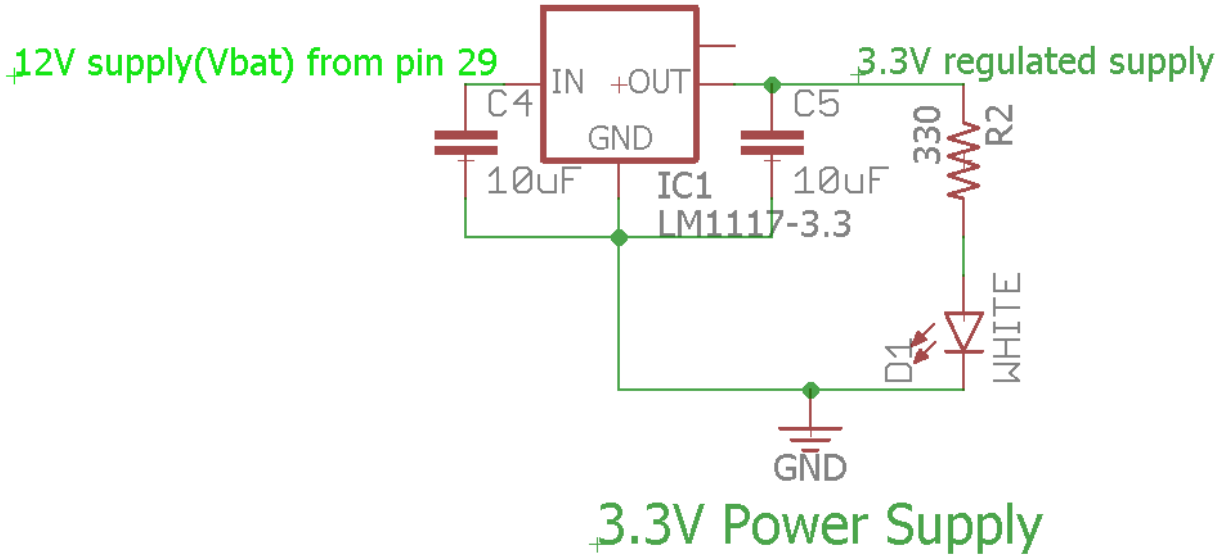
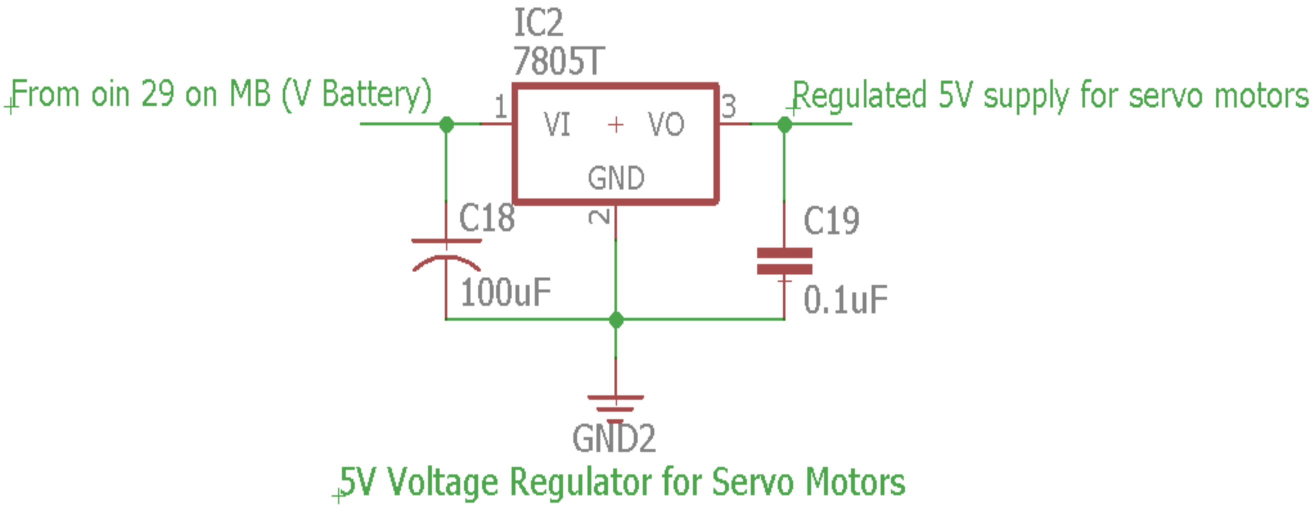


figure 1.

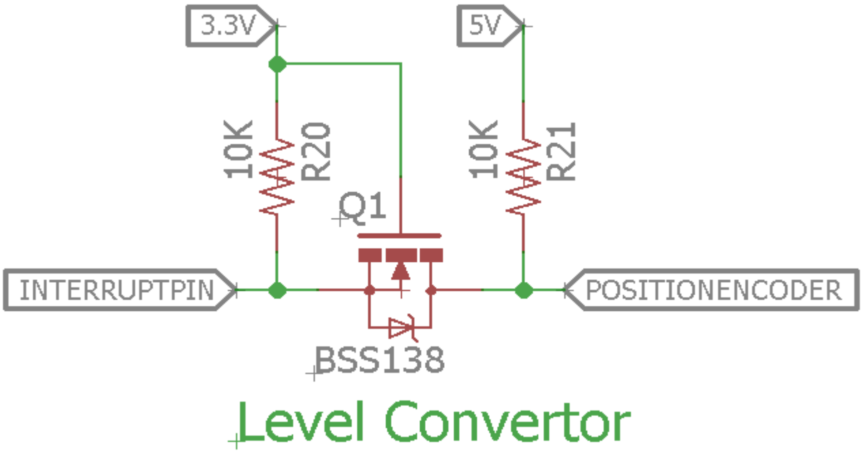
5.1.2 Powering Servos

The servo motors can operate safely up to 6V, beyond this voltage they get damaged. Also, the servos require high current. There is a separate power line for servos taken from Pin 29 and reduced to 5V using the voltage regulator. Refer the schematic for further details.


IC2
7805T
From oin 29 on MB (V Battery)
1
VI
+
VO
3
GND
2
Regulated 5V supply for servo motors
C18
100uF
C19
0.1uF
GND2
5V Voltage Regulator for Servo Motors

5.2 Level Converters

The TIVA platform operates at 3.3V and the Firebird operates at 5V. Directly connecting these pins to the TIVA may be fatal. So to interface these sensors, a level converter is used. A bidirectional MOSFET based level converter used. The level converter is necessary is for input pins. In the boards Level converter is used for interfacing the position encoders of the motors. Refer the schematic for further details.



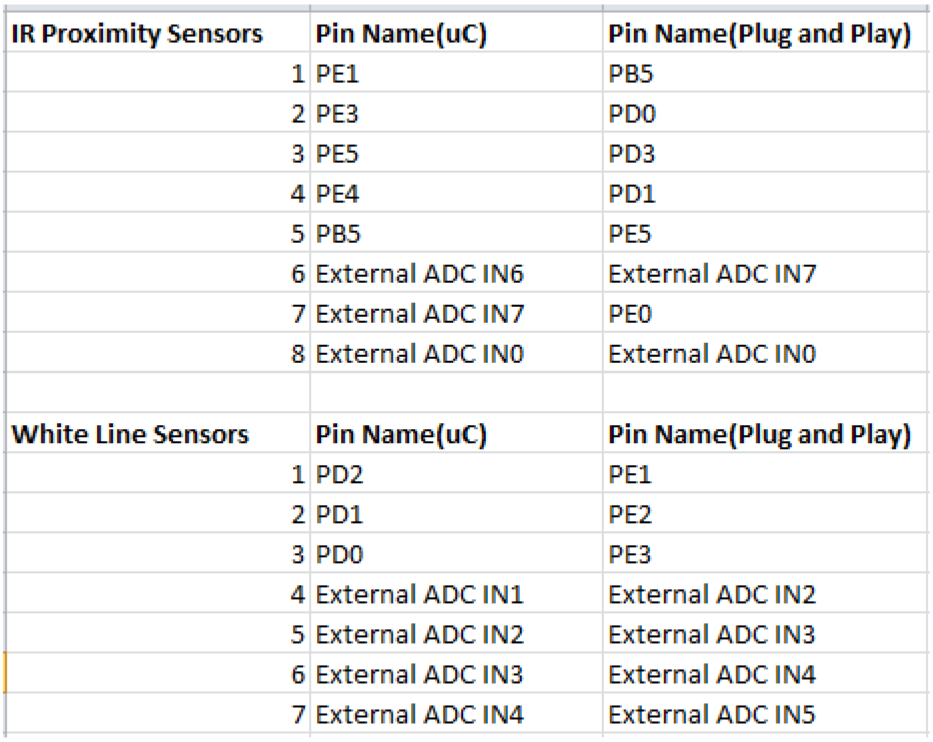
NOTE: If the user wishes to interface extra sensors using the GPIOs provided o then external level converters have to used if the output of the sensor is ab

5.3 Sensors

The firebird V has as many as 22 sensors, but maximum 12 sensors can be interfaced directly with the controller. The daughter board has interfaced 20 of those 22 sensors using external I2C bases ADC. Sensors that were not included in the daughter board are current sensor and battery monitoring sensor. These sensors are working either on 3.3V or on 5V. Interfacing 3.3V sensors are simple and can be directly connected to the controller. On the other hand 5V can not be directly interfaced so a different approach is taken which will be mentioned in the 5V sensors sub heading.

5.3.1 3.3V sensors

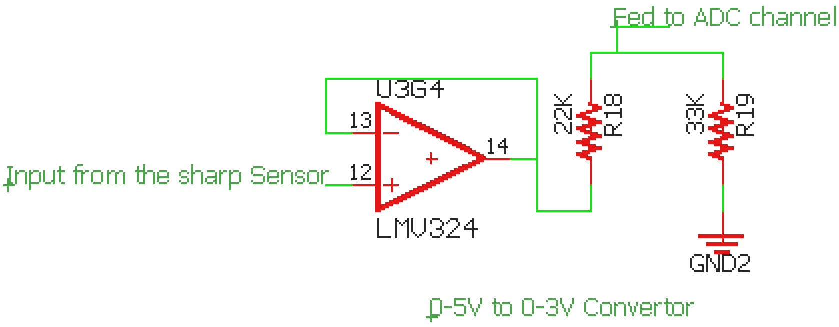
The output white line sensors and IR Proximity sensors vary from 0 to 3.3V. Hence these sensors can be interfaced directly with the microcontroller. Refer the table below for pin connections.

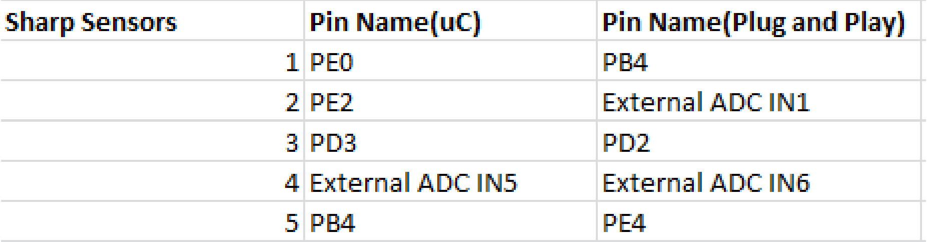


IR Proximity Sensors	Pin Name(uC)	Pin Name(Plug and Play)
1	PE1	PB5
2	PE3	PD0
3	PE5	PD3
4	PE4	PD1
5	PB5	PE5
6	External ADC IN6	External ADC IN7
7	External ADC IN7	PE0
8	External ADC IN0	External ADC IN0
White Line Sensors	**Pin Name(uC)**	**Pin Name(Plug and Play)**
1	PD2	PE1
2	PD1	PE2
3	PD0	PE3
4	External ADC IN1	External ADC IN2
5	External ADC IN2	External ADC IN3
6	External ADC IN3	External ADC IN4
7	External ADC IN4	External ADC IN5

5.3.2 5V sensors

Sharp Sensors are the only sensors on board that works on 5V supply. The output of the sharp sensor ranges from 0-5V and according to the output we have a formula to calculate the distance. While uC has VREF as 3.3V so these sensors cannot be directly connected. The approach we followed is to feed the output of the sensor to a buffer and then using a voltage divider convert 0-5 range to 0-3V range. For better understanding refer to the schematic below. There is a also table which tells about the pin connection.

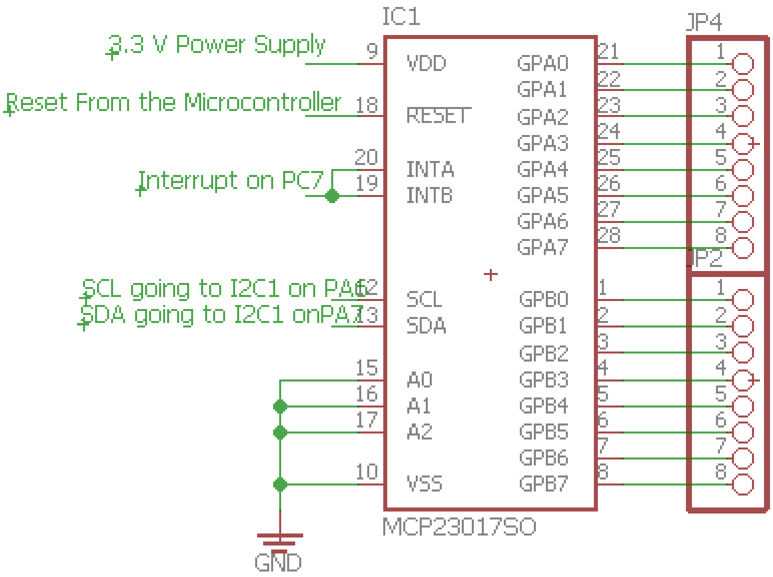




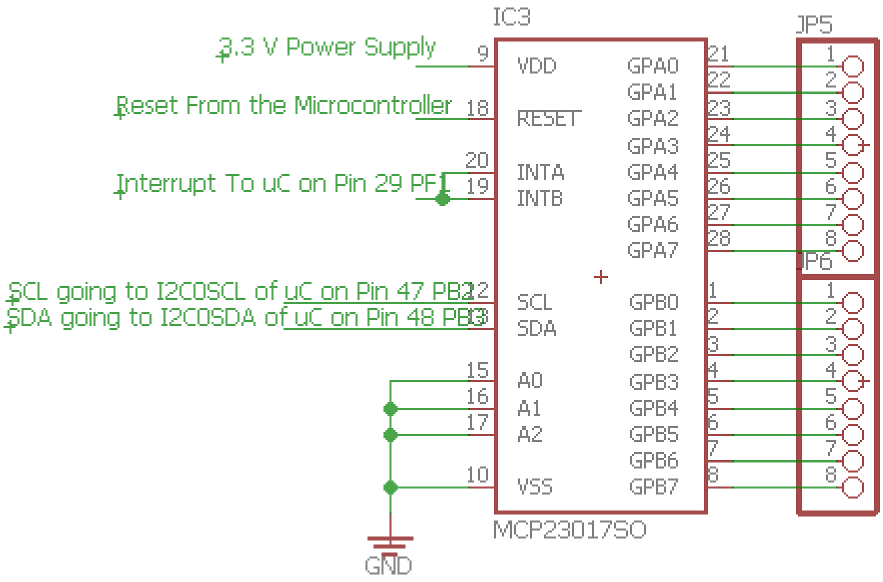
Sharp Sensors	Pin Name(uC)	Pin Name(Plug and Play)
1	PE0	PB4
2	PE2	External ADC IN1
3	PD3	PD2
4	External ADC IN5	External ADC IN6
5	PB4	PE4

5.4 Port Expander

TM4C123GH6PM has only 64 pins out which only 43 are GPIO pins. This limits our application to read input and respond correspondingly. To increase the number of GPIO and there interrupts we have used I2C compatible a port expander MCP23017. It has 2 PORTS A and B, with each port having 8 Pins.The interrupts on each pin can also be monitored. To read more about it, download the datasheet from here.The schematic of the connection is shown below.Keep in mind that I2C SCL and SDA have already been pulled up using 10K resistor.



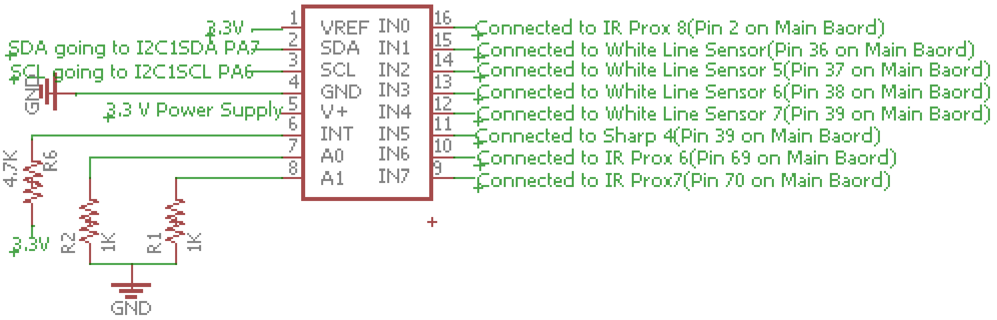
Port Expander For Plug and Play



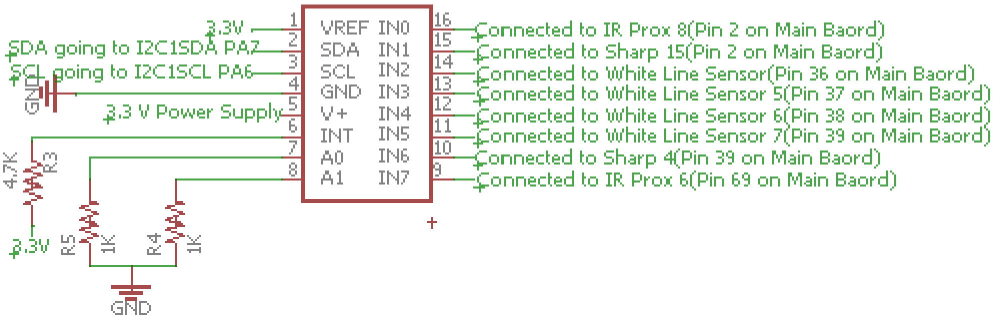
Port Expander For uC

5.5 External ADC

It has already been mentioned that adc channels on the microcontroller is limited to 12 while firebird has 22 sensors available. We have interfaced an external ADC which is also I2C compatible. It has 8 channel with 12 bit resolution.To read more about it, download the datasheet from here.The schematic of the connection is shown below.Keep in mind that I2C SCL and SDA have already been pulled up using 10K resistor.


3.3V
SDA going to I2C1SDA PA7
SCL going to I2C1SCL PA6
GND
3.3 V Power Supply
1
2
3
4
5
6
7
8
VREF
SDA
SCL
GND
V+
INT
A0
A1
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7
16
15
14
13
12
11
10
9
Connected to IR Prox 8(Pin 2 on Main Baord)
Connected to White Line Sensor(Pin 36 on Main Baord)
Connected to White Line Sensor 5(Pin 37 on Main Baord)
Connected to White Line Sensor 6(Pin 38 on Main Baord)
Connected to White Line Sensor 7(Pin 39 on Main Baord)
Connected to Sharp 4(Pin 39 on Main Baord)
Connected to IR Prox 6(Pin 69 on Main Baord)
Connected to IR Prox7(Pin 70 on Main Baord)
4.7K
R6
3.3V
R2
1K
R1
1K
GND

External ADC connection for uC Board


3.3V
SDA going to I2C1SDA PA7
SCL going to I2C1SCL PA6
GND
3.3 V Power Supply
1
2
3
4
5
6
7
8
VREF
SDA
SCL
GND
V+
INT
A0
A1
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7
16
15
14
13
12
11
10
9
Connected to IR Prox 8(Pin 2 on Main Baord)
Connected to Sharp 15(Pin 2 on Main Baord)
Connected to White Line Sensor(Pin 36 on Main Baord)
Connected to White Line Sensor 5(Pin 37 on Main Baord)
Connected to White Line Sensor 6(Pin 38 on Main Baord)
Connected to White Line Sensor 7(Pin 39 on Main Baord)
Connected to Sharp 4(Pin 39 on Main Baord)
Connected to IR Prox 6(Pin 69 on Main Baord)
4.7K
R3
3.3V
R5
1K
R4
1K
GND

External ADC connections for Plug and Play Board

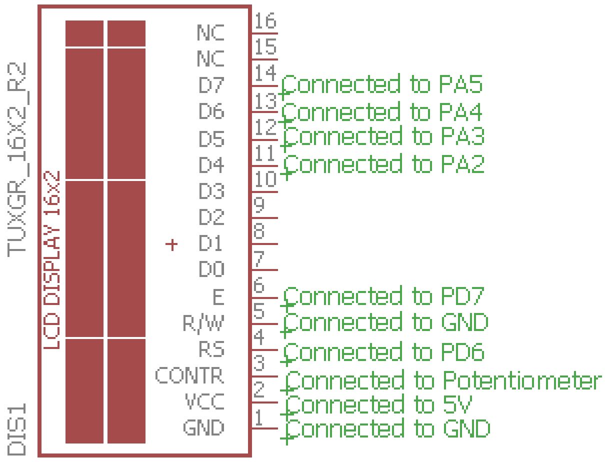
5.6 LCD Interfacing

LCD can be interfaced in 8bit or 4 bit interfacing mode. In 8 bit mode it requires 3 control line and 8 data lines. To reduce number of I/Os required, Fire Bird V robot uses 4 bit interfacing mode which requires 2 control lines and 4 data lines. In this mode upper and lower nibble of the data/command byte needs to be sent separately. RW(Read/Write) control line of lcd is grounded so it can only work in write mode.

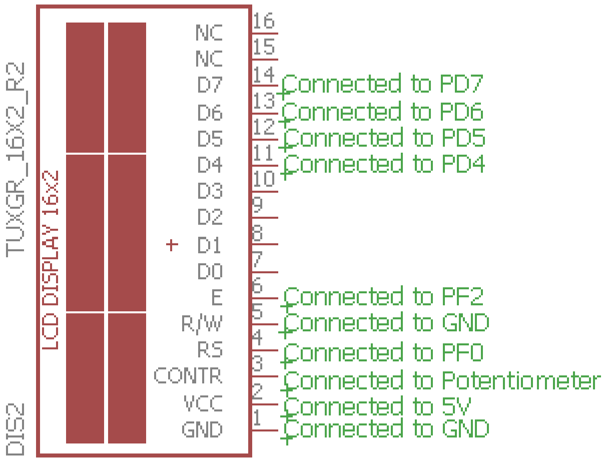
The EN line is used to tell the LCD that microcontroller has sent data to it or microcontroller is ready to receive data from LCD. This is indicated by a high-to-low transition on this line. To send data to the LCD, program should make sure that this line is low (0) and then set the other two control lines as required and put data on the data bus. When this is done, make EN high (1) and wait for the minimum amount of time as specified by the LCD datasheet, and end by bringing it to low (0) again.

When RS is low (0), data is treated as a command or special instruction by the LCD

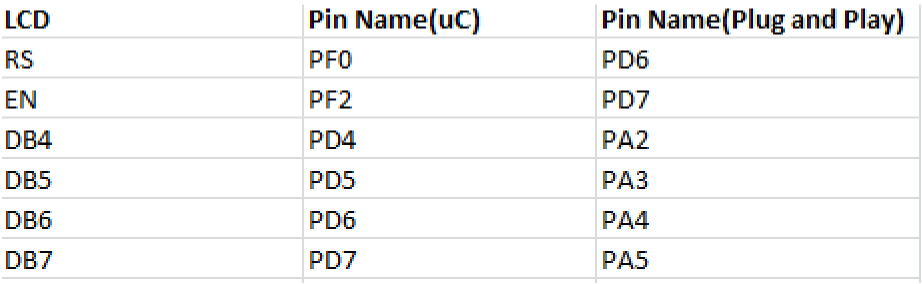
(such as clear screen, position cursor, etc.). When RS is high (1), data being sent is treated as text data which should be displayed on the screen.
written to the LCD.//



LCD Connections of Plug and Play board



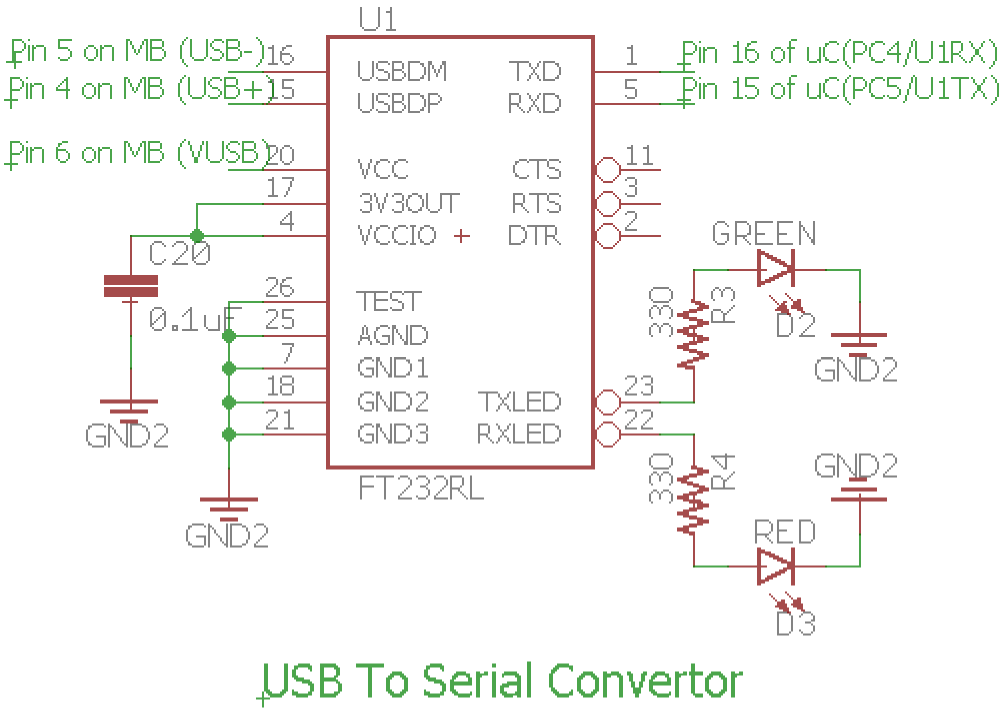
LCD Connections of uC based board



LCD	Pin Name(uC)	Pin Name(Plug and Play)
RS	PF0	PD6
EN	PF2	PD7
DB4	PD4	PA2
DB5	PD5	PA3
DB6	PD6	PA4
DB7	PD7	PA5

5.7 USB Communication

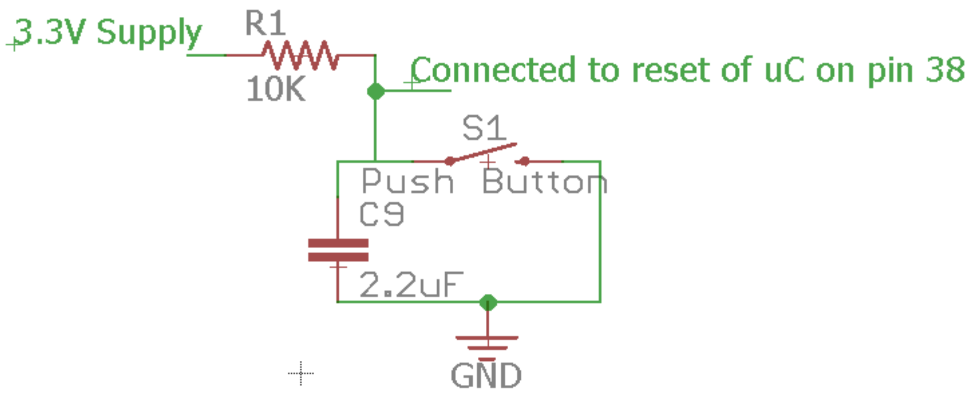
Fire Bird V's main board has USB port socket. Microcontroller accesses USB port via main board socket. All its pins are connected to the microcontroller adapter board via main board's socket connector.FT232 is a USB to TTL level serial converter. It is used for adding USB connectivity to the microcontroller adapter board. With onboard USB circuit Fire Bird V can communicate serially with the PC through USB port without the use of any external USB to Serial converter. Microcontroller socket uses USB port from the main board. Data transmission and reception is indicated suing TX and RX LEDs which are located near the FT232 IC. This IC is only on the uC based board. Plug and play board has its own usb port on TIVA launcpad. The schematic of ft232 is shown below.



5.8 Programing the Controller

5.9 Reset Switch

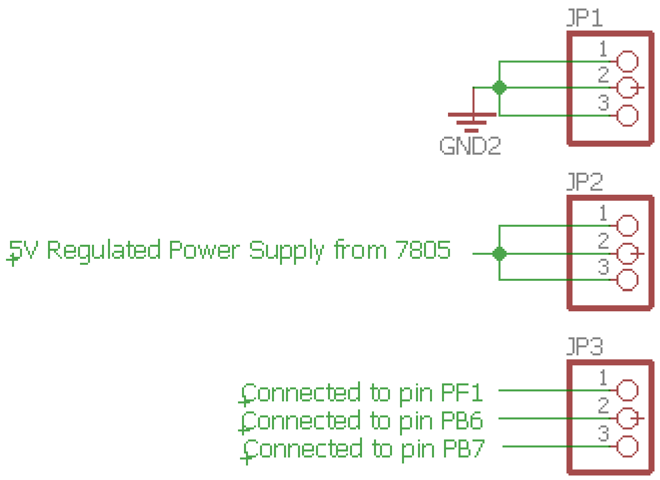
The Plug and play board makes use of reset button present on the TIVA launchpad. The uC based has a switch connected to the reset the reset pin 38 of the microcontroller. The schematic is given below.



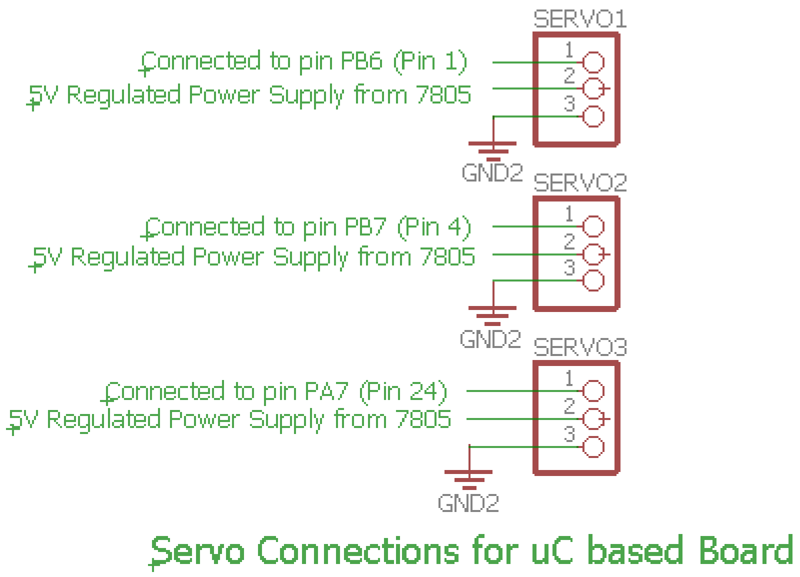
Reset Switch

5.10 Servo Connectors

The microcontroller board has three Servo connectors. It can be used for driving servo motors of camera pod or any other attachment. Power for the servo connector is provided by the "5V servo supply" voltage regulator. Both the board have different pwm pins for servo which can be seen from the schematic.



Servo Connections for Plug and Play Board


SERVO1
Connected to pin PB6 (Pin 1)
5V Regulated Power Supply from 7805
1
2
3
GND2
SERVO2
Connected to pin PB7 (Pin 4)
5V Regulated Power Supply from 7805
1
2
3
GND2
SERVO3
Connected to pin PA7 (Pin 24)
5V Regulated Power Supply from 7805
1
2
3
GND2
Servo Connections for uC based Board

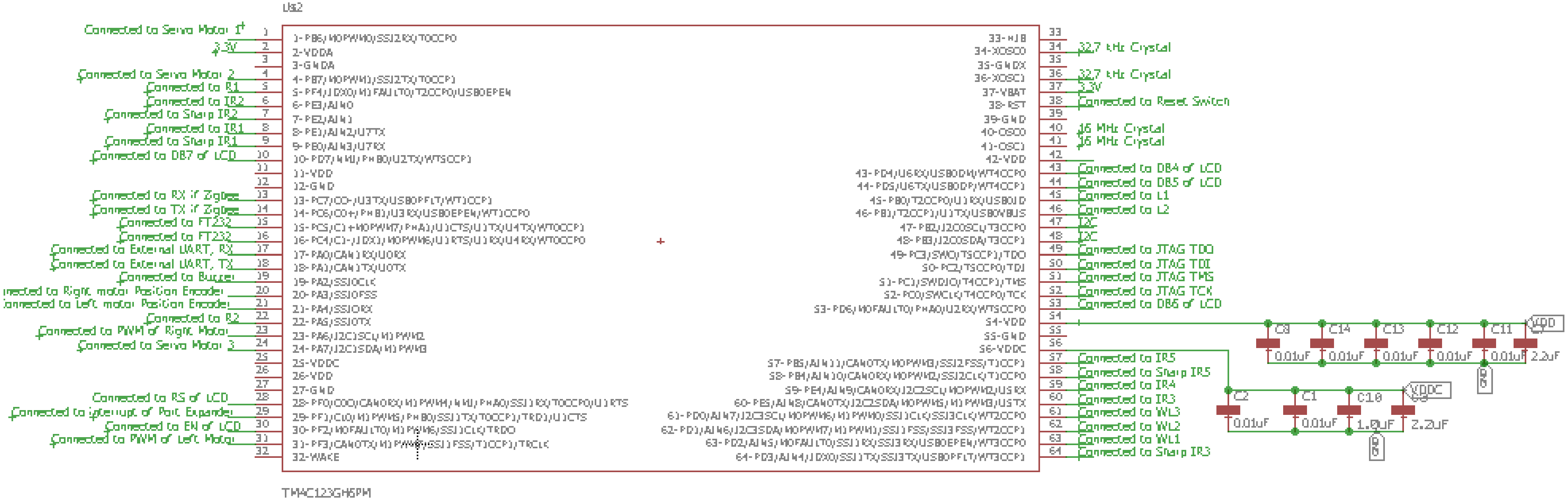
5.11 TM4C123GH6PM Micro-controller:

The Tiva^{TM} C Series ARM Cortex-M4 microcontrollers provide top performance and advanced integration. The product family is positioned for cost-conscious applications requiring significant control processing and connectivity capabilities such as:

- Low power, hand-held smart devices
- Gaming equipment
- Home and commercial site monitoring and control
- Motion control
- Medical instrumentation
- Test and measurement equipment
- Factory automation
- Fire and security
- Smart Energy/Smart Grid solutions
- Intelligent lighting control
- Transportation

Schematic Of the connections is shown below.

5.11 TM4C123GH6PM Micro-controller:


U$2
TM4C123GH6PM
Connected to Servo Motor 1
3.3V
Connected to Servo Motor 2
Connected to R1
Connected to IR2
Connected to Sharp IR2
Connected to IR1
Connected to Sharp IR1
Connected to DB7 of LCD
Connected to RX if Zigbee
Connected to TX if Zigbee
Connected to FT232
Connected to FT232
Connected to External UART, RX
Connected to External UART, TX
Connected to Buzzer
nected to Right motor Position Encoder
onnected to Left motor Position Encoder
Connected to R2
Connected to PWM of Right Motor
Connected to Servo Motor 3
Connected to RS of LCD
Connected to interupt of Port Expander
Connected to EN of LCD
Connected to PWM of Left Motor
1-PB6/M0PWM0/SSI2RX/T0CCP0
2-VDDA
3-GNDA
4-PB7/M0PWM1/SSI2TX/T0CCP1
5-PF4/IDX0/M1FAULT0/T2CCP0/USB0EPEN
6-PE3/AIN0
7-PE2/AIN1
8-PE1/AIN2/U7TX
9-PE0/AIN3/U7RX
10-PD7/NMI/PhB0/U2TX/WT5CCP1
11-VDD
12-GND
13-PC7/C0-/U3TX/USB0PFLT/WT1CCP1
14-PC6/C0+/PhB1/U3RX/USB0EPEN/WT1CCP0
15-PC5/C1+M0PWM7/PhA1/U1CTS/U1TX/U4TX/WT0CCP1
16-PC4/C1-/IDX1/M0PWM6/U1RTS/U1RX/U4RX/WT0CCP0
17-PA0/CAN1RX/U0RX
18-PA1/CAN1TX/U0TX
19-PA2/SSI0CLK
20-PA3/SSI0FSS
21-PA4/SSI0RX
22-PA5/SSI0TX
23-PA6/I2C1SCL/M1PWM2
24-PA7/I2C1SDA/M1PWM3
25-VDDC
26-VDD
27-GND
28-PF0/C0O/CAN0RX/M1PWM4/NMI/PhA0/SSI1RX/T0CCP0/U1RTS
29-PF1/C1O/M1PWM5/PhB0/SSI1TX/T0CCP1/TRD1/U1CTS
30-PF2/M0FAULT0/M1PWM6/SSI1CLK/TRD0
31-PF3/CAN0TX/M1PWM7/SSI1FSS/T1CCP1/TRCLK
32-WAKE
33-HIB
34-XOSC0
35-GNDX
36-XOSC1
37-VBAT
38-RST
39-GND
40-OSC0
41-OSC1
42-VDD
43-PD4/U6RX/USB0DM/WT4CCP0
44-PD5/U6TX/USB0DP/WT4CCP1
45-PB0/T2CCP0/U1RX/USB0ID
46-PB1/T2CCP1/U1TX/USB0VBUS
47-PB2/I2C0SCL/T3CCP0
48-PB3/I2C0SDA/T3CCP1
49-PC3/SWO/T5CCP1/TDO
50-PC2/T5CCP0/TDI
51-PC1/SWDIO/T4CCP1/TMS
52-PC0/SWCLK/T4CCP0/TCK
53-PD6/M0FAULT0/PhA0/U2RX/WT5CCP0
54-VDD
55-GND
56-VDDC
57-PB5/AIN11/CAN0TX/M0PWM3/SSI2FSS/T1CCP1
58-PB4/AIN10/CAN0RX/M0PWM2/SSI2CLK/T1CCP0
59-PE4/AIN9/CAN0RX/I2C2SCL/M0PWM4/U5RX
60-PE5/AIN8/CAN0TX/I2C2SDA/M0PWM5/M1PWM3/U5TX
61-PD0/AIN7/I2C3SCL/M0PWM6/M1PWM0/SSI1CLK/SSI3CLK/WT2CCP0
62-PD1/AIN6/I2C3SDA/M0PWM7/M1PWM1/SSI1FSS/SSI3FSS/WT2CCP1
63-PD2/AIN5/M0FAULT0/SSI1RX/SSI3RX/USB0EPEN/WT3CCP0
64-PD3/AIN4/IDX0/SSI1TX/SSI3TX/USB0PFLT/WT3CCP1
32.7 kHz Crystal
32.7 kHz Crystal
3.3V
Connected to Reset Switch
16 MHz Crystal
16 MHz Crystal
Connected to DB4 of LCD
Connected to DB5 of LCD
Connected to L1
Connected to L2
I2C
I2C
Connected to JTAG TDO
Connected to JTAG TDI
Connected to JTAG TMS
Connected to JTAG TCK
Connected to DB6 of LCD
Connected to IR5
Connected to Sharp IR5
Connected to IR4
Connected to IR3
Connected to WL3
Connected to WL2
Connected to WL1
Connected to Sharp IR3
C8 0.01uF
C14 0.01uF
C13 0.01uF
C12 0.01uF
C11 0.01uF
2.2uF
VDD
GND
C2 0.01uF
C1 0.01uF
C10 1.0uF
2.2uF
VDDC
GND

5.12 Pin Functionality

5.12.1 Pin of uC

Pin No.	Pin	Complete Pin Connections
1	PB6	Servo Motor 1
2	VDDA	VDD filtered through capcitors
3	GNDA	Ground
4	PB7	Servo Motor 2
5	PF4	Pin 53 Right Motor 1
6	PE3	16 IR proximity Sensor 2
7	PE2	Output of Second OpAmp of lm324
8	PE1	12 IR Proximity Sensor 1
9	PE0	output of First OpAmp of lm324
10	PD7	28 DB7 of LCD Data
11	VDD	VDD filtered through capcitors
12	GND	Ground
13	PC7	13 Zigbee Rx
14	PC6	14 Zigbee Tx
15	PC5	FT232
16	PC4	FT232
17	PA0	External UART
18	PA1	External UART
19	PA2	Buzzer
20	PA3	63 Right Position Encoder Interrupt
21	PA4	64 Left Position Encoder Interrupt
22	PA5	55 Right Motor 2
23	PA6	54 Right Motor Pwm
24	PA7	Servo Motor 3
25	VDDC	Connected to VDDC on 56
26	VDD	VDD filtered through capcitors
27	GND	Ground
28	PF0	22 RS of LCD
29	PF1	INT A and B of to GPIO expander shorted and connected
30	PF2	24 EN of LCD
31	PF3	50 Left Motor PWM
32	WAKE	Ground
33	HIB	NC
34	XOSC0	32.7 KHz crystals(One End)

35	GNDX	Cap to crystal
36	XOSC1	32.7 KHz crystals(Other End)
37	VBAT	3.3 Volts
38	RST	Reset Switch
39	GND	Ground
40	OSC1	16 MHz crystal(One end)
41	OSC1	16 MHz crystal(Other end)
42	VDD	VDD filtered through capcitors
43	PD4	26 DB4 Of LCD
44	PD5	25 DB5 of LCD
45	PB0	51 Left Motor 1
46	PB1	52 Left Motor 2
47	PB2	I2C ADC SCL
48	PB3	I2C ADC SDA
49	PC3	JTAG TDO
50	PC2	JTAG TDI
51	PC1	JTAG TMS
52	PC0	JTAG TCK
53	PD6	27 DB6 Of LCD
54	VDD	VDD filtered through capcitors
55	GND	Ground
56	VDDC	Connected to VDDC on 25
57	PB5	46 IR Proximity Sensor 5
58	PB4	Output of First OpAmp of lm358
59	PE4	43 IR Proximity Sensor 4
60	PE5	42 IR Proximity Sensor 3
61	PD0	32 White Line Sensor 3
62	PD1	31 White Line Sensor 2
63	PD2	30 White Line Sensor 1
64	PD3	Output of Third OpAmp of lm324

5.12.2 Robot Main Board Connections

Pin Out	Pin Name	Functionality	PIN on DB	Pin on Pluggable B
1	Current sensor	Current sense analog value	Not Using	

2	IR Proximity sensor 8	IR Proximity sensor 8 analog value	External Adc INT0	
3	GND	Ground	Ground	
4	DATA+	USB connection going to the ATMEGA2560 USB connection with uC	C4	
5	DATA-	microcontroller via FT232 USB to serial USB connection with uC	PC5	
6	VCC	USB converter. Connect TO VCC of FT232		
7	5V System	”5V System Voltage. Can be used for powering up any digital device with current limit of 400mA.”		
8	5V System	”5V System Voltage. Can be used for powering up any digital device with current limit of 300mA.”		
9	5V System	”5V System Voltage. Can be used for powering up any digital device with current limit of 300mA.”		
10	5V System	”5V System Voltage. Can be used for powering up any digital device with current limit of 400mA.”		
11	SHARP IR Range Sensor 1	Analog output of Sharp IR range Sensor 1	PE0(lm324 1)	
12	IR Proximity Sensor 1	Analog output of IR Proximity sensor 1	PE1	
13	XBee RXD	XBee wireless module Serial data in	PC7	
14	XBee TXD	XBee wireless module Serial data out	PC6	
15	SHARP IR Range Sensor 2	Analog output of Sharp IR range sensor 2	PE2(lm324 2)	
16	IR Proximity Sensor 2	Analog output of IR Proximity sensor 2	PE3	
17	RSSI	To capture the RSSI signal		
18	MOSI	MOSI of the Controller/NC create extra expansion headers		
19	MISO	MISO of controller/NC create extra expansion headers		

20	SCK	SCK of the controller/NC create extra expansion headers		
21	SSI	SS of the controller/ NC create extra expansion headers		
22	RS	connected to RS of LCD normal I/O	PF0	
23	RW	connected to RW of LCD normal I/O	GND	
24	EN	connected to EN of LCD normal I/O	PF2	
25	DB5	data pin of lcd normal I/O	PD5	
26	DB4	data pin of lcd normal I/O	PD4	
27	DB6	data pin of lcd normal I/O	PD6	
28	DB7	data pin of lcd normal I/O	PD7	
29	V Battery System	ADC to check the level of battery voltage		
30	WL1	Analog output of white line sensor 1	PD2	
31	WL2	Analog output of white line sensor 2	PD1	
32	WL3	Analog output of white line sensor 3	PD0	
33	”Sharp IR Sensors 1and 5 Disable”			
34	IR Proximity Sensor Disable			
35	5V System	”5V system Voltage. Can be used for powering	up any digital device. Current Limit: 400mA.”	
36	WL4	Analog output of white line sensor 4	External Adc INT1	
37	WL5	Analog output of white line sensor 5	External Adc INT2	

38	WL6	Analog output of white line sensor 6	External Adc INT3	
39	WL7	Analog output of white line sensor 7	External Adc INT4	
40	White Line Sensors Disable			
41	Sharp IR Range Finder 3	Analog output of Sharp IR range sensor 3	PD3 (lm 324 3)	
42	IR Proximity Sensor 3	Analog output of IR Proximity sensor 3	PE5	
43	IR Proximity Sensor 4	Analog output of IR Proximity sensor 4	PE4	
44	Sharp IR Range Finder 4	Analog output of Sharp IR range sensor 4	in 5 ex (lm 324 5)	
45	Sharp IR Range Finder 5	Analog output of Sharp IR range sensor 5	PB4 (lm358 1)	
46	IR Proximity Sensor 5	Analog output of IR Proximity sensor 5	PB5	
47	C11	motor not present		
48	C1	PWM not present		
49	c12	not present		
50	PWM L	left motor PWM(timer pin in PWM mode)	PF3	
51	L1	left motor pin1 normal I/O	PB0	
52	L2	left motor pin2 normal I/O	PB1	
53	R1	right motor pin1 normal I/O	PF4	
54	PWM R2	right motor PWM(timer pin in PWM mode)	PA6	
55	R2	right motor pin2	PA5	
56	NC			
57	NC			
58	NC			
59	NC			
60	NC			
61	NC			

62	Position encoder left	Output of Left position encoder (0-5V) PA4		
63	Position encoder right	Output of Right position encoder (0-5V) PA3		
64	position enocder C2	Output of C2 position encoder (0-5V)		
65	Position encoder C1	Output of C1 position encoder (0-5V)		
66	C22	NC		
67	C21	NC		
68	C2	Pwm	NC	
69	IR Prox6	Analog output of IR Proximity sensor 6 External Adc	INT6	
70	IR Prox7	Analog output of IR Proximity sensor 7 External Adc	INT7	
71	Buzzer	Input, V¿0.65V turns on the Buzzer	PA2	
72	DAC Out	NC		
73	RS232 TX	NC		
74	RS232 RX	NC		

5.12.3 Pin Connection Of Plug And Play Board

Pin Name	Pin Connection on Main Board	Function
PA0		Used for Programming
PA1		Used for Programming
PA2	26	DB4 of LCD
PA3	27	DB5 of LCD
PA4	28	DB6 of LCD
PA5	29	DB7 of LCD
PA6		I2C
PA7		I2C
PB0	14	Zigbee Tx
PB1	13	Zigbee RX
PB2	62	Position encoder of left motor
PB3	52	L2
PB4	11	Sharp IR1
PB5	12	IR 1
PB6		Servo
PC0		

PC1		
PC2		
PC3		
PC4	53	R1
PC5	54	PWM of right motor
PC6	55	R2
PC7		Interrupt of port expander
PD0	16	IR 2
PD1	43	IR Prox 4
PD2	41	Sharp IR 3
PD3	42	IR Prox 3
PD4		
PD5		
PD6	22	RS of LCD
PD7	24	EN of LCD
PE0	70	IR 7
PE1	30	WL1
PE2	31	WL2
PE3	32	WL3
PE4	45	Sharp IR 5
PE5	46	IR 5
PE6		
PE7		
PF0	63	Position encoder of right motor
PF1		Servo
PF2	50	PWM of left motor
PF3	51	L1
PF4	71	Buzzer

6 Software Manual:

6.1 Code Composer Studio:

Code Composer Studio is an integrated development environment (IDE) that supports TI's Microcontroller and Embedded Processors portfolio. Code Composer Studio comprises a suite of tools used to develop and debug embedded applications. It includes an optimizing C/C++ compiler, source code editor, project build environment, debugger, profiler, and many other features. The intuitive IDE provides a single user interface taking you through each step of the application development flow. Familiar tools and interfaces allow users to get started faster than ever before. Code Composer Studio combines the advantages of the Eclipse software framework with advanced embedded debug capabilities from TI resulting in a compelling feature-rich development environment for embedded developers. This description is directly taken from the website of Texas Instruments and click to know more about CC Studio

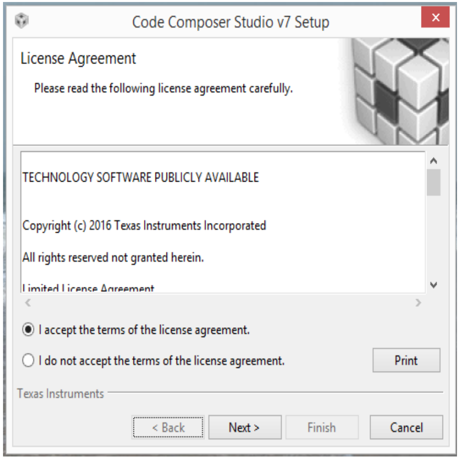
6.1.1 Download CC Studio:

At the time of writing this document Version 7 was the latest one. You can check for the latest at Download CCS.(do not download any beta versions).There will be two installer files.The web installer will require Internet access until it completes. If the web installer version is unavailable or you can't get it to work, download, unzip and run the offline version. The offline download will be much larger than the installed size of CCS since it includes all the possible supported hardware.

6.1.2 Installing C C Studio:

After the installer has started follow the steps mentioned below:

1. Accept the Software License Agreement and click Next.



Code Composer Studio v7 Setup

License Agreement

Please read the following license agreement carefully.

TECHNOLOGY SOFTWARE PUBLICLY AVAILABLE

Copyright (c) 2016 Texas Instruments Incorporated

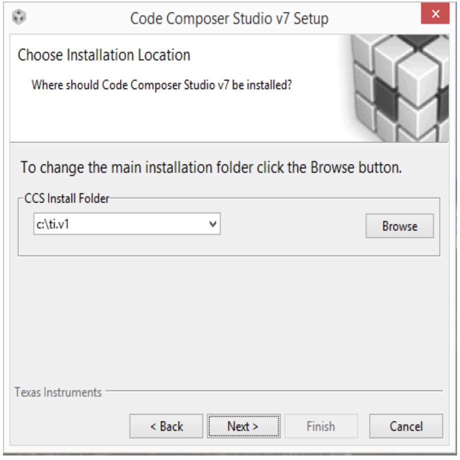
All rights reserved not granted herein.

Limited License Agreement

● I accept the terms of the license agreement.

○ I do not accept the terms of the license agreement.

2. Select the destination folder and click next.



Code Composer Studio v7 Setup

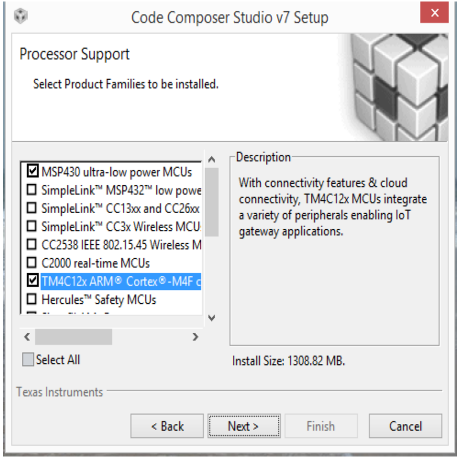
Choose Installation Location

Where should Code Composer Studio v7 be installed?

To change the main installation folder click the Browse button.

CCS Install Folder: c:\ti.v1

3. Select the processors that your CCS installation will support. You must select ”TM4C12X Arm Cortex M4”. You can select other architectures, but the installation time and size will increase.



Code Composer Studio v7 Setup

Processor Support

Select Product Families to be installed.

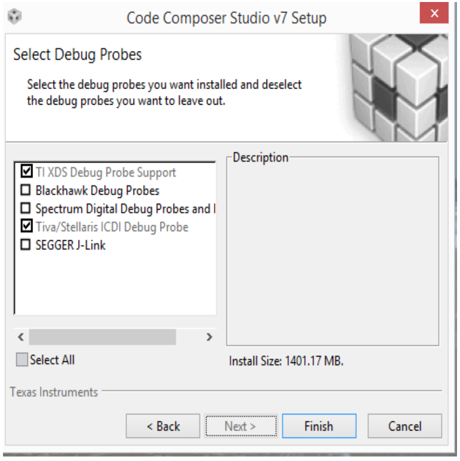
- ☑ MSP430 ultra-low power MCUs
- □ SimpleLink™ MSP432™ low powe
- □ SimpleLink™ CC13xx and CC26xx
- □ SimpleLink™ CC3x Wireless MCU
- □ CC2538 IEEE 802.15.45 Wireless M
- □ C2000 real-time MCUs
- ☑ TM4C12x ARM® Cortex®-M4F c
- □ Hercules™ Safety MCUs

□ Select All

Description: With connectivity features & cloud connectivity, TM4C12x MCUs integrate a variety of peripherals enabling IoT gateway applications.

Install Size: 1308.82 MB.

4. Select debug probes and click finish



Code Composer Studio v7 Setup

Select Debug Probes

Select the debug probes you want installed and deselect the debug probes you want to leave out.

- ☑ TI XDS Debug Probe Support
- □ Blackhawk Debug Probes
- □ Spectrum Digital Debug Probes and I
- ☑ Tiva/Stellaris ICDI Debug Probe
- □ SEGGER J-Link

□ Select All

Description

Install Size: 1401.17 MB.

5. The installer process should take 15 - 30 minutes, depending on the speed of your connection. The offline installation should take 10 to 15 minutes. When the installation is complete, uncheck the “Launch Code Composer Studio v7” checkbox and then click Finish.There are several additional tools that require installation during the CCS install process. Click “Yes” or “OK” to proceed when these appear.

6. Install TivaWare for C Series (Complete). Download and install the latest full version of TivaWare from: TivaWare. The filename is SW-TM4C-x.x.exe . This workshop was built using version 1.1. Your version may be a later one. If at all possible, please install TivaWare into the default location.

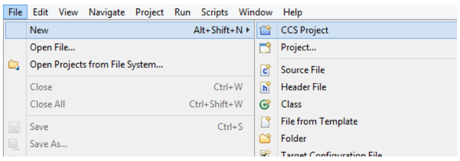
You can find additional information at these websites:
Main page: www.ti.com/launchpad
Tiva C Series TM4C123G LaunchPad:
http://www.ti.com/tool/ek-tm4c123gxl
TM4C123GH6PM folder:
http://www.ti.com/product/tm4c123gh6pm
BoosterPack webpage: www.ti.com/boosterpack
LaunchPad Wiki:
www.ti.com/launchpadwiki

For understanding the launchpad properly and to learn more about Tiva it is strongly recommended to go through the webpage TIva Worshops and download and read the workbook

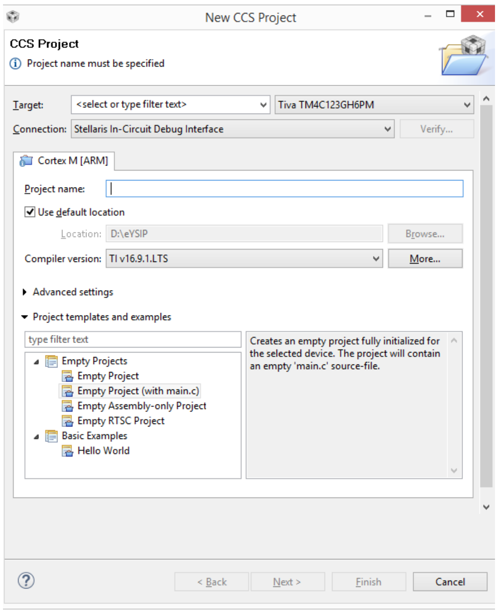
6.1.3 Create a New Project

To create new project follow the steps mentioned:

1. Click File then New and then CCS Projects



2. Select Target and connection as shown in the photo. Give a name to your project and save in a location.Click Finish. A main.c file will be open



New CCS Project

CCS Project

Project name must be specified

Target: <select or type filter text> Tiva TM4C123GH6PM

Connection: Stellaris In-Circuit Debug Interface

Cortex M [ARM]

Project name:

☑ Use default location

Location: D:\eYSIP

Compiler version: TI v16.9.1.LTS

Creates an empty project fully initialized for the selected device. The project will contain an empty 'main.c' source-file.

6.1.4 Add Path and Build Variables

The path and build variables are used for:

- Path variable – when you ADD (link) a file to your project, you can specify a ”relative to” path. The default is PROJECT_LOC which means that your linked resource (like a .lib file) will be linked relative to your project directory.

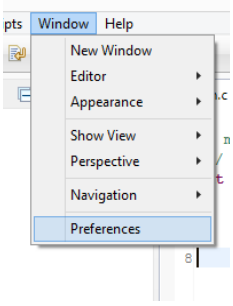
- Build variable – used for items such as the search path for include files associated with a library – i.e. it is used when you build your project.

Variables can either have a PROJECT scope (that they only work for this project) or a WORKSPACE scope (that they work across all projects in the workspace). In the next step, we need to add (link) a library file and then add a search path for include files. First, we'll add these variables MANUALLY as WORKSPACE variables so that any project in your workspace can use the variables. Refer to the workbook by TI for adding as PROJECT

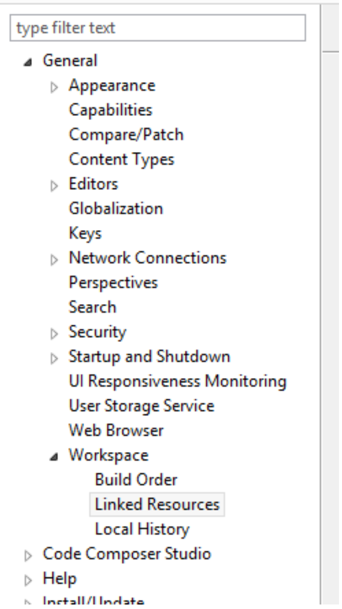
6.1.4.1 Adding a Path Variable

To add a path variable,:

- Right-click on your Window Tab and select Preference.



- Expand General list in the upper left-hand corner as shown and then expand the Resource list and click on Linked Resources: We want to add a New variable to specify exactly where you installed TivaWare.

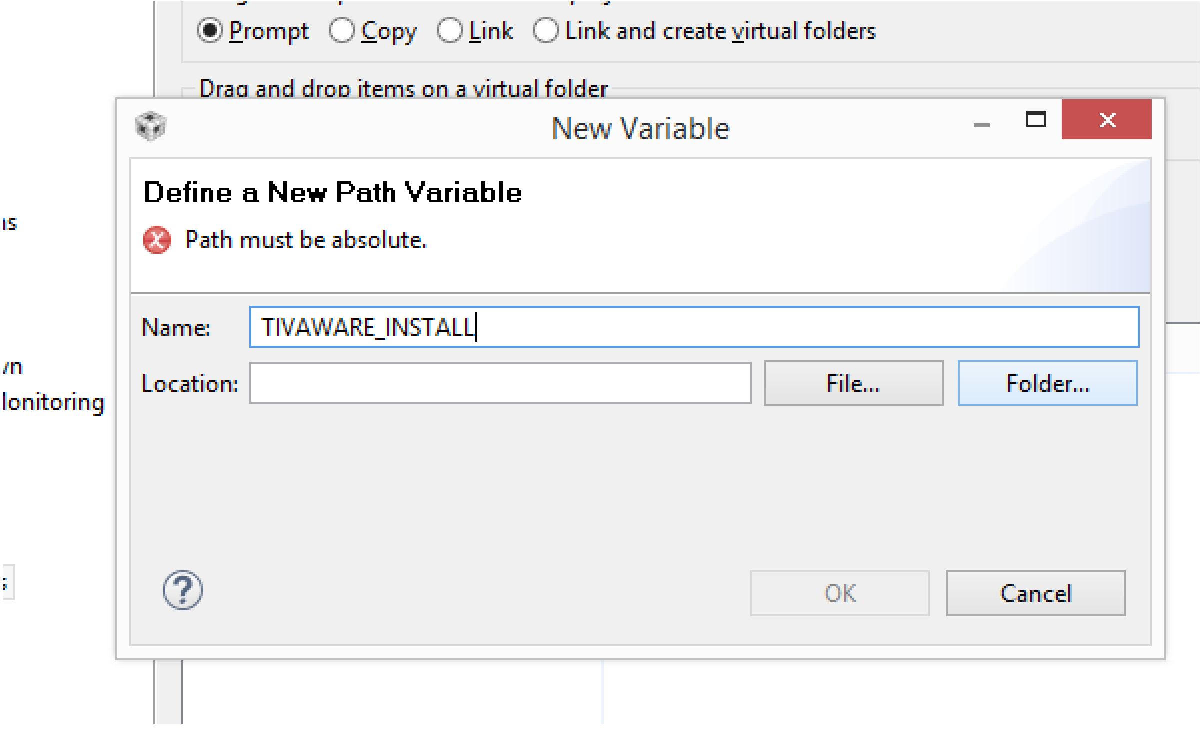


type filter text

- General
  - Appearance
  - Capabilities
  - Compare/Patch
  - Content Types
  - Editors
  - Globalization
  - Keys
  - Network Connections
  - Perspectives
  - Search
  - Security
  - Startup and Shutdown
  - UI Responsiveness Monitoring
  - User Storage Service
  - Web Browser
  - Workspace
    - Build Order
    - Linked Resources
    - Local History
- Code Composer Studio
- Help
- Install/Update

- Click New

- When the New Variable dialog appears, type TIVAWARE_INSTALL for the name.



● Prompt ○ Copy ○ Link ○ Link and create virtual folders

Drag and drop items on a virtual folder

New Variable

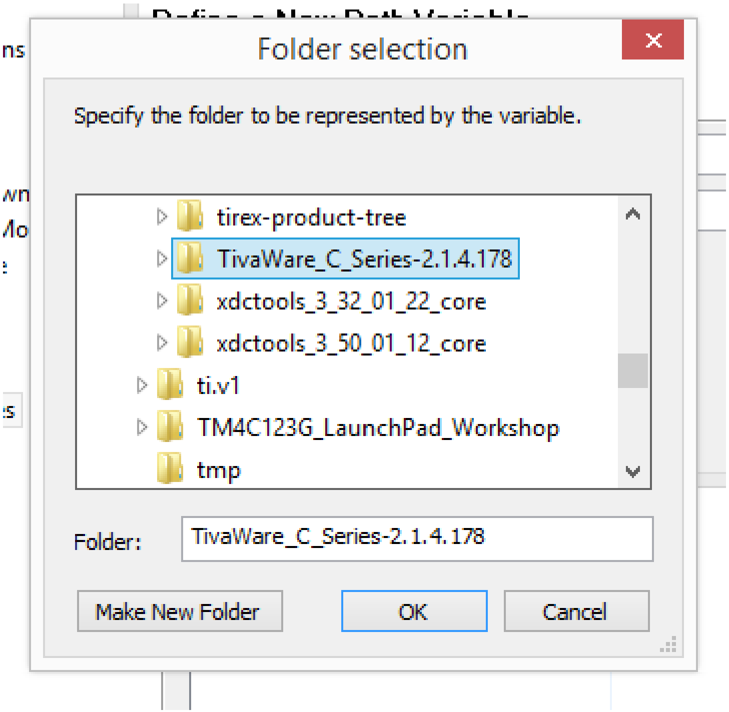
Define a New Path Variable

Path must be absolute.

Name: TIVAWARE_INSTALL

Location:

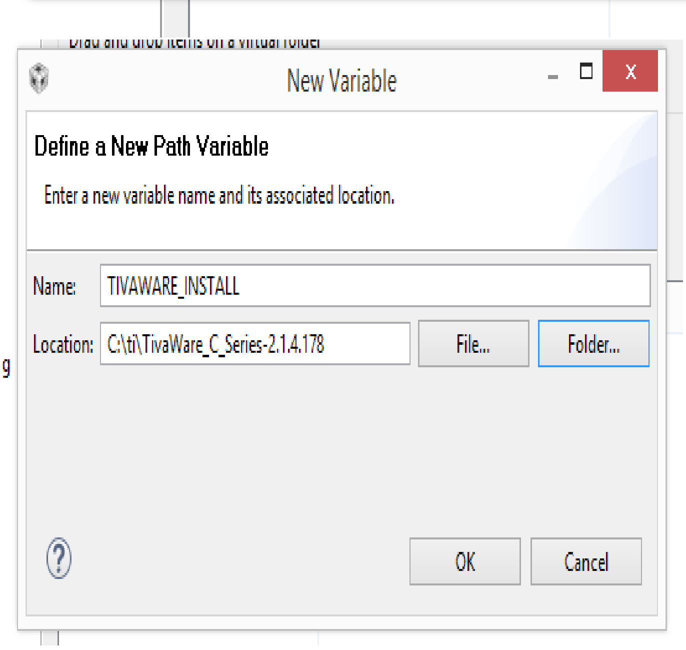
- For the Location, click the Folder... button and navigate to your TivaWare installation. Click on the folder name and then click OK.



Folder selection

Specify the folder to be represented by the variable.

Folder: TivaWare_C_Series-2.1.4.178



New Variable

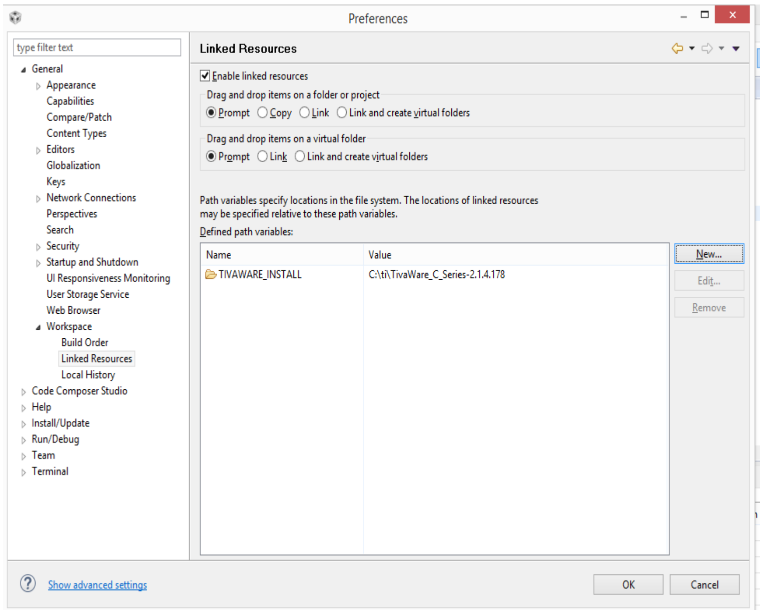
Define a New Path Variable

Enter a new variable name and its associated location.

Name: TIVAWARE_INSTALL

Location: C:\ti\TivaWare_C_Series-2.1.4.178

- Click OK. You should see your new variable listed in the Variables list.



Linked Resources

☑ Enable linked resources

Drag and drop items on a folder or project

● Prompt ○ Copy ○ Link ○ Link and create virtual folders

Drag and drop items on a virtual folder

● Prompt ○ Link ○ Link and create virtual folders

Path variables specify locations in the file system. The locations of linked resources may be specified relative to these path variables.

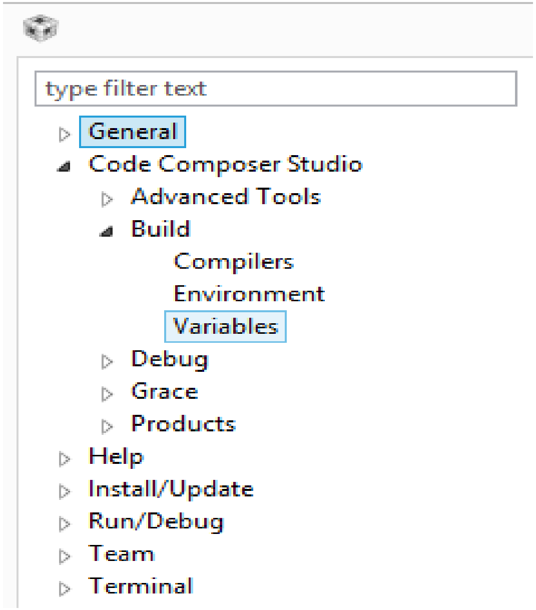
Defined path variables:

Name	Value
TIVAWARE_INSTALL	C:\ti\TivaWare_C_Series-2.1.4.178

6.1.4.2 Adding a Build Variable

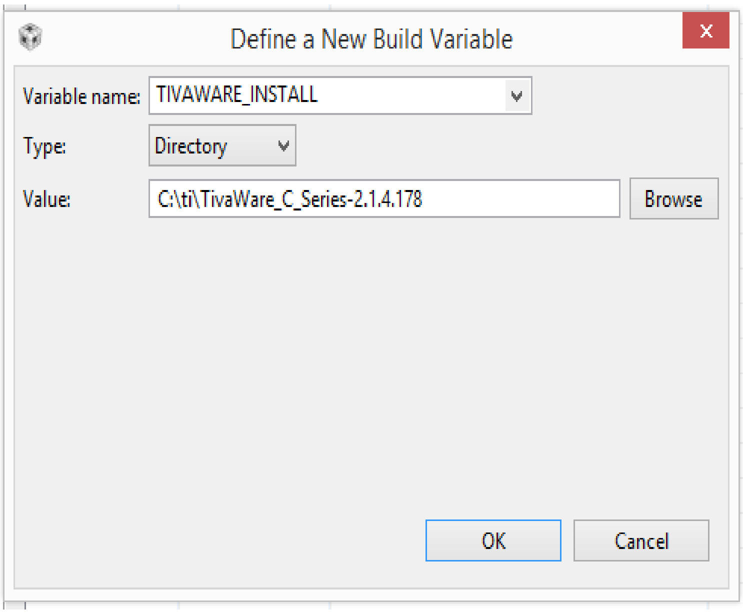
Now let's add a build variable that we will use in the include search path for the INCLUDE files associated with the TivaWare driver libraries.

- Click on Code Composer Studio Build and then the Variables tab:



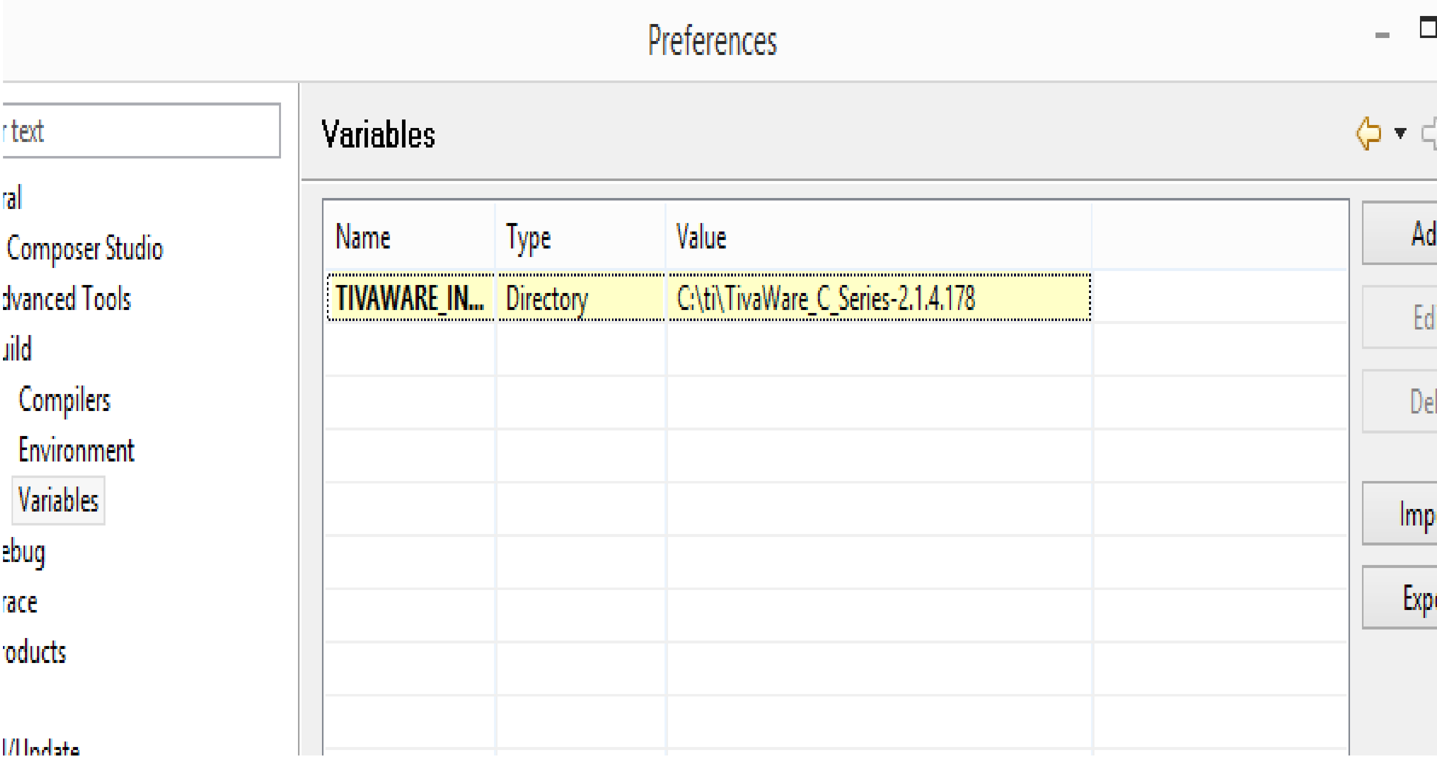
- Click the Add button. When the Define a New Build Variable dialog appears, insert TIVAWARE_INSTALL into the Variables name box.

- Change the Type to Directory and browse to your Tivaware installation folder.



Variable name: TIVAWARE_INSTALL
Type: Directory
Value: C:\ti\TivaWare_C_Series-2.1.4.178

- Click OK.
- Click OK again to save and close the Build Properties window.



Name	Type	Value
TIVAWARE_IN...	Directory	C:\ti\TivaWare_C_Series-2.1.4.178

6.2 Buzzer

Located in the folder "Buzzer_Beep" folder in the documentation. In this example, we will load buzzer beep code in Tiva based Fire Bird V. Now we will see in detail the structure of this code.This experiment demonstrates the simple operation of Buzzer ON/OFF with one second delay. Buzzer is connected to PORTF 4 pin of the Tiva Launchpad. If you have uC based board then it is connected to PORTA 2.

6.2.1 Code

```
    #include <stdint.h>
    #include <stdbool.h>
    #include "inc/hw_types.h"
    #include "inc/hw_memmap.h"
    //This header File is important to Unlock GPIO Pins
    #include "inc/hw_gpio.h"
    #include "driverlib/sysctl.h"
    #include "driverlib/gpio.h"

    /**** Useful Macros Definition******/
    /******Remove the comments if you are using uC board******
    #define buzzerEnable     SYSCTL_PERIPH_GPIOA
    #define buzzer           GPIO_PORTA_BASE
    #define buzzerPin        GPIO_PIN_2
    *****************************************************/

    /******Remove the comments if you are using uC board******/
    #define buzzerEnable     SYSCTL_PERIPH_GPIOF
    #define buzzer           GPIO_PORTF_BASE
    #define buzzerPin        GPIO_PIN_4
    /*****************************************************/

    #define buzzerOn()       GPIOPinWrite(buzzer,buzzerPin,255)
    #define buzzerOff()      GPIOPinWrite(buzzer,buzzerPin,0)
    /*********************************/
    void setupCLK();
    void peripheralEnable();
    void configIOPin();
    void delay_ms(uint64_t delay);
    void delay_ms(uint64_t delay);

    int main(void) {
      setupCLK();
      peripheralEnable();
      configIOPin();
      while(1){
        buzzerOn();
        delay_ms(1000);
        buzzerOff();
        delay_ms(1000);
      }
    }
    /*****************************************************************************
    * This function is used to setup Clock frequency of the controller
    * It can be changed through codes
    * In this we have set frequency as 40Mhz
    * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
    *****************************************************************************/
    void setupCLK(){
      SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
    }
    /******************************
    * Enabling System Peripherals
```

```
* buzzer Port in this case
* buzzerPin for buzzer output
*****************************/
void peripheralEnable(){
  SysCtlPeripheralEnable(buzzerEnable);
  /***** Just in case you are not familiar with macros*****
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
  *************This is enabling PORTF********************/
}
/***********************************
* Configuring Pin as Input Or Output
***********************************/
void configIOPin(){
  GPIOPinTypeGPIOOutput(buzzer, buzzerPin);
  /***** Just in case you are not familiar with macros*****
  GPIOPinTypeGPIOOutput(buzzer, GPIO_PIN_4);
  ****************This is P4 output********************/
}
/******************************************
* Calculating Delays
* extern void SysCtlDelay(uint32_t ui32Count)
* waits until the counting has been completed
******************************************/
void delay_ms(uint64_t delay){
  SysCtlDelay(delay*SysCtlClockGet()/3000.0);
}
void delay_us(uint64_t delay){
  SysCtlDelay(delay*SysCtlClockGet()/3000000.0);
}
```

6.3 Simple I/O Operation

This experiment demonstrates the simple I/O operations. This example is only for plug and play board, but this should not discourage the user from understanding the example as it provide very important example of I/O operation on TIVA platform.

In this lab you have to use switch SW1, SW2 and RGB LED present on Tiva C series board.

1. Use switch SW1 to Turn on Red LED on first switch press, Green LED on second switch press and Blue LED on third switch press. Repeat the same cycle next switch press onwards. Note that LED should remain on for the duration switch is kept pressed i.e. LED should turn off when switch is released.

2. Use switch SW2 and sw2Status (a variable). Your program should increment sw2Status by one, every time switch is pressed. Note how the value of sw2Status changes on each switch press. Use debugger and add sw2Status to Watch Expression" window. (You will find Continuous Refresh button on top of the Expression Window). You can use step debugging or breakpoints to check the variable value. Hint:To add variable to Expression Window, select and right click the variable name and select Add Watch Expression". To view Expression Window, click on View button from CCS menu bar and select Expressions.

6.3 Simple I/O Operation

6.3.1 Code for Plug and Play Board

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_gpio.h" //To unlock locked pins for GPIO
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#define userSwitch1 GPIO_PIN_0
#define redLed      GPIO_PIN_1
#define blueLed     GPIO_PIN_2
#define greenLed    GPIO_PIN_3
#define userSwitch2 GPIO_PIN_4
#define LOCK_F (*((volatile unsigned long *)0x40025520))
#define CR_F   (*((volatile unsigned long *)0x40025524))
void setupCLK();
void configIOPin();
void delay_ms(uint64_t delay);
void delay_us(uint64_t delay);

int main(){
  setupCLK();
  SysCtlDelay(3);
  configIOPin();
  unsigned char pinData=1;
  unsigned char state=2;
  unsigned char countSwitch2=0;
  unsigned char flagSW1=0;
  unsigned char flagSW2=0;
  while(1){
    pinData=GPIOPinRead(GPIO_PORTF_BASE,userSwitch2|userSwitch1);
    if((pinData&0x01)==0)
      flagSW1=1;
    else if((flagSW1==1)&&(pinData&0x01)==1){
      countSwitch2+=1;
      flagSW1=0;
    }
    if((pinData&0x10)==0){
      GPIOPinWrite(GPIO_PORTF_BASE,redLed|blueLed|greenLed,state);
      flagSW2=1;
    }
    else if(((flagSW2==1)&&(pinData&0x10)==0x10)){
      flagSW2=0;
      GPIOPinWrite(GPIO_PORTF_BASE,redLed|blueLed|greenLed,0);
      state=state*2;
      if(state>8)
        state=2;
    }
    delay_ms(5);
  }
}
/*
***********************************************************************************
* This function is used to setup Clock frequency of the controller
* Enabling System Peripherals
* PORTF in this case
*****************************/
void setupCLK(){
  SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN)
;
          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
        }
        /*********************************************
        * Configuring Pin as Input Or Output
        * PF0 by default is locked and cannot
        * be used as input unless it is unlocked
        *********************************************/
        void configIOPin(){
          GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, redLed|blueLed|greenLed);
          HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
          HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
          HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;
          GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, userSwitch2|userSwitch1);
          GPIOPadConfigSet(GPIO_PORTF_BASE ,userSwitch2|userSwitch1 ,GPIO_STRENGTH_12MA,
GPIO_PIN_TYPE_STD_WPU);
        }
        /**********************************
        * Calculating Delays
        **********************************/
        void delay_ms(uint64_t delay){
          SysCtlDelay(delay*(SysCtlClockGet()/3000));
        }
        void delay_us(uint64_t delay){
          SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
        }

```

6.4 Robot Direction Control

Located in the folder "Experiments Motion Control Simple" folder. Robot's motors are controlled by L293D motor controller. Using L293D, microcontroller can control direction and velocity of both of the motors. To change the direction appropriate logic levels (High/Low) are applied to IC L293D's direction pins. Velocity control is done using pulse width modulation (PWM) applied to Enable pins of L293D IC.

The Motor connections are as shown below

6.4.1 Code for Plug and Play Board:

```
        #include <stdint.h>
        #include <stdbool.h>
        #include "inc/hw_types.h"
        #include "inc/hw_memmap.h"
        //This header File is important to Unlock GPIO Pins
        #include "inc/hw_gpio.h"
        #include "driverlib/sysctl.h"
        #include "driverlib/gpio.h"
        //Used for controlling Motor direction
        #define right           0x41
        #define left            0x18
        #define softRight       0x10
        #define softLeft        0x01
```

```
#define forward          0x11
#define backward         0x48

void setupCLK();
void peripheralEnable();
void configIOPin();
void delay_ms(uint64_t delay);
void delay_us(uint64_t delay);
void motion(uint8_t);

int main(void) {
  setupCLK();
  peripheralEnable();
  configIOPin();
  while(1){
    motion(forward);
    delay_ms(1000);
    motion(right);
    delay_ms(1000);
    motion(left);
    delay_ms(1000);
    motion(backward);
    delay_ms(1000);
  }
}
/*
********************************************************************************
* This function is used to setup Clock frequency of the controller
* It can be changed through codes
* In this we have set frequency as 40Mhz
* Frequency is set by SYSDIV which can be found in data sheet for different
frequencies

********************************************************************************
*/
void setupCLK(){
  SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN)
;
}
/*****************************
* Enabling System Peripherals
* PORTF,PORTB and PORTC in this case
*****************************/
void peripheralEnable(){
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
}
/***********************************
* Configuring Pin as Input Or Output
* And Setting PWM Pin to Always High
***********************************/
void configIOPin(){
  GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_3);
  GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE,GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6);
  GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_3|GPIO_PIN_2);
  GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_2,255);
  GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_5,255);
}
/***********************************
```

```
* Calculating Delays
*************************************/
void delay_ms(uint64_t delay){
  SysCtlDelay(delay*(SysCtlClockGet()/3000));
}
void delay_us(uint64_t delay){
  SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
}
/***************************************************
* This function is for giving the direction of motion
* Macros have been defined at starting
* Macros for directions are 8 bits
* Out of these 8 bits only 4 are used
* Bit 0 (LSB) corresponds to PB3
* Bit 3       corresponds to PF3
* Bit 4       corresponds to PC4
* Bit 6       corresponds to PF6
**************************************************/
void motion(uint8_t direction){
  GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_3,direction<<3);
  GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_4|GPIO_PIN_6,direction);
  GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3,direction);
}

```

6.4.2 Code for uC based Board:

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
//This header File is important to Unlock GPIO Pins
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
//Used for control Motor direction
#define right         0x22
#define left          0x11
#define softRight     0x02
#define softLeft      0x10
#define forward       0x12
#define backward      0x21
#define stop          0x00

void setupCLK();
void peripheralEnable();
void configIOPin();
void delay_ms(uint64_t delay);
void delay_ms(uint64_t delay);
void motion(uint8_t);

int main(void) {
  setupCLK();
  peripheralEnable();
  configIOPin();
  while(1){
    motion(forward);
    delay_ms(1000);
    motion(right);
    delay_ms(1000);
    motion(left);
    delay_ms(1000);
    motion(backward);
    delay_ms(1000);
  }
}
/*************************************************************************************
* This function is used to setup Clock frequency of the controller
* It can be changed through codes
* In this we have set frequency as 40Mhz
* Frequency is set by SYSDIV which can be found in data sheet for different frequencies
**************************************************************************************/
void setupCLK(){
  SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
}
/*******************************
* Enabling System Peripherals
* PORTF,PORTB and PORTC in this case
******************************/
void peripheralEnable(){
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
}
/*************************************
* Configuring Pin as Input Or Output
* And Setting PWM Pin to Always High
*************************************/
void configIOPin(){
  GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_0|GPIO_PIN_1);
  GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE,GPIO_PIN_6|GPIO_PIN_5);
  GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_4|GPIO_PIN_3);
  GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3,255);
  GPIOPinWrite(GPIO_PORTA_BASE,GPIO_PIN_6,255);
}
/*************************************
* Calculating Delays
*************************************/
void delay_ms(uint64_t delay){
  SysCtlDelay(delay*(SysCtlClockGet()/3000));
}
void delay_us(uint64_t delay){
  SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
}
/*****************************************************
* This function is for giving the direction of motion
* Macros have been defined at starting
* Macros for directions are 8 bits
* Out of these 8 bits only 4 are used
* Bit 0 (LSB) corresponds to PB0
* Bit 1       corresponds to PF1
* Bit 4       corresponds to PF4
* Bit 5       corresponds to PA5
*****************************************************/
void motion(uint8_t direction){
  GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_0|GPIO_PIN_1,direction);
```

```
        GPIOPinWrite(GPIO_PORTA_BASE,GPIO_PIN_5,direction);
        GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_4,direction);
    }

```

6.5 Robot Position Control Using Interrupts

Position encoders give position / velocity feedback to the robot. It is used in closed loop to control robot's position and velocity. Position encoder consists of optical encoder and slotted disc assembly. When this slotted disc moves in between the optical encoder we get square wave signal whose pulse count indicates position and time period indicates velocity.

Connections:

Plug and Play Board:

PB2 : External interrupt for left motor position encoder

PF0 : External interrupt for the right position encoder

uC Based Board:

PA4 : External interrupt for left motor position encoder

PA3 : External interrupt for the right position encoder

6.5.1 Calculation of position encoder resolution:

To be added later

6.5.2 Code for Plug and Play Board:

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
//This header File is important to Unlock GPIO Pins
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"

#define right          0x41
#define left           0x18
#define softRight      0x10
#define softLeft       0x01
#define forward        0x11
#define backward       0x48
#define stop           0x00

```

```
void setupCLK();
void peripheralEnable();
void configIOPin();
void delay_ms(uint64_t delay);
void delay_us(uint64_t delay);
void motion(uint8_t);
void interruptEnable();
void encoderInterruptEncountered();
void encoderInterruptEncountered1();
void angleRotate(uint16_t Degrees);
void linearDistanceMM(unsigned int DistanceInMM);
void rightDegrees(unsigned int Degrees);
void leftDegrees(unsigned int Degrees);
void forwardMM(unsigned int DistanceInMM);
void backwardMM(unsigned int DistanceInMM);
volatile unsigned long int ShaftCountRight = 0;
volatile unsigned long int ShaftCountLeft = 0;

int main(void) {
  setupCLK();
  peripheralEnable();
  configIOPin();
  interruptEnable();
  while(1){
    forwardMM(100);
    delay_ms(1000);
    leftDegrees(90);
    delay_ms(1000);
  }
}
/*
**********************************************************************************
* This function is used to setup Clock frequency of the controller
* It can be changed through codes
* In this we have set frequency as 40Mhz
* Frequency is set by SYSDIV which can be found in data sheet for different
frequencies

**********************************************************************************
*/
void setupCLK(){
  SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN)
;
}
/******************************
* Enabling System Peripherals
* PORTF,PORTB and PORTC in this case
******************************/
void peripheralEnable(){
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
}
/*************************************
* Configuring Pin as Input Or Output
* Unlocking PF0
* Setting PWM Pins to Always High
* Weak Pull to the Input Pins
*************************************/
void configIOPin(){
      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_3);
      GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE,GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6);
      GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1);
      GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_2,255);
      GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_5,255);
      HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
      HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
      HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;
      GPIOPinTypeGPIOInput(GPIO_PORTF_BASE, GPIO_PIN_0);
      GPIOPadConfigSet(GPIO_PORTF_BASE,GPIO_PIN_0,GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);
      GPIOPinTypeGPIOInput(GPIO_PORTB_BASE, GPIO_PIN_2);
      GPIOPadConfigSet(GPIO_PORTB_BASE,GPIO_PIN_2,GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);
    }
    /************************************
    * Calculating Delays
    ************************************/
    void delay_ms(uint64_t delay){
      SysCtlDelay(delay*(SysCtlClockGet()/3000));
    }
    void delay_us(uint64_t delay){
      SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
    }
    /***************************************************
    * This function is for giving the direction of motion
    * Macros have been defined at starting
    * Macros for directions are 8 bits
    * Out of these 8 bits only 4 are used
    * Bit 0 (LSB) corresponds to PB3
    * Bit 3       corresponds to PF3
    * Bit 4       corresponds to PC4
    * Bit 6       corresponds to PF6
    ***************************************************/
    void motion(uint8_t direction){
      GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_3,direction<<3);
      GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_4|GPIO_PIN_6,direction);
      GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3,direction);
      }
      /****For Enabling Interrupt on PORTF and PORTB****/
      void interruptEnable(){
      GPIOIntDisable(GPIO_PORTF_BASE,GPIO_PIN_0);
      GPIOIntClear(GPIO_PORTF_BASE,GPIO_PIN_0);
      GPIOIntRegister(GPIO_PORTF_BASE, encoderInterruptEncountered);
      GPIOIntTypeSet(GPIO_PORTF_BASE,GPIO_PIN_0,GPIO_FALLING_EDGE);
      GPIOIntEnable(GPIO_PORTF_BASE,GPIO_PIN_0);
      GPIOIntDisable(GPIO_PORTB_BASE,GPIO_PIN_2);
      GPIOIntClear(GPIO_PORTB_BASE,GPIO_PIN_2);
      GPIOIntRegister(GPIO_PORTB_BASE, encoderInterruptEncountered1);
      GPIOIntTypeSet(GPIO_PORTB_BASE,GPIO_PIN_2,GPIO_FALLING_EDGE);
      GPIOIntEnable(GPIO_PORTB_BASE,GPIO_PIN_2);
    }
    /**** ISR For External Interrupt on PortF***********************
    * Check on which pin of the PORTA has encountered an interrupt
    * There is only one ISR for complete PORT
    * No two PORTs can have same ISR
    **************************************************************/
    void encoderInterruptEncountered(){
      if(GPIOIntStatus(GPIO_PORTF_BASE, false)&GPIO_PIN_0){
        ShaftCountRight++;
        GPIOIntClear(GPIO_PORTF_BASE, GPIO_PIN_0);
      }
      if(GPIOIntStatus(GPIO_PORTB_BASE, false)&GPIO_PIN_2){
        ShaftCountLeft++;
        GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_1,2);
        GPIOIntClear(GPIO_PORTB_BASE, GPIO_PIN_2);
      }
    }
    /***************************************************
    * Function to Rotate to desired Angle
    * Resolution can be Change to Get Higher Precision
    ***************************************************/
    void angleRotate(uint16_t Degrees){
      unsigned long int ReqdShaftCountInt = 0;
      ReqdShaftCountInt = Degrees/ 4.09;// division by resolution to get shaft count
      ShaftCountRight = 0;
      while (1){
        if((ShaftCountRight>=ReqdShaftCountInt))
          break;
      }
      motion(stop);
    }
    /***************************************************
    * Function to Move in a Linear Distance
    * Resolution can be Change to Get Higher Precision
    ***************************************************/
    void linearDistanceMM(unsigned int DistanceInMM){
      unsigned long int ReqdShaftCountInt = 0;
      ReqdShaftCountInt =DistanceInMM / 5.338;;
      ShaftCountRight=0;
      ShaftCountLeft=0;
      while(1){
        if((ShaftCountRight >=ReqdShaftCountInt)&&(ShaftCountLeft >= ReqdShaftCountInt))
          break;
        else if((ShaftCountRight > ReqdShaftCountInt))
          motion(softRight);
        else if((ShaftCountLeft > ReqdShaftCountInt))
          motion(softLeft);
      }
      motion(stop); //Stop robot
    }
    void forwardMM(unsigned int DistanceInMM){
      motion(forward);
      linearDistanceMM(DistanceInMM);
    }

    void backwardMM(unsigned int DistanceInMM){
      motion(backward);
      linearDistanceMM(DistanceInMM);
    }
    void leftDegrees(unsigned int Degrees){
      motion(left); //Turn left
      angleRotate(Degrees);
    }
    void rightDegrees(unsigned int Degrees){
      motion(right); //Turn right
      angleRotate(Degrees);
    }

```

6.5.3 Code for uC based Board:

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/tm4c123gh6pm.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"

#define right           0x22
#define left            0x11
#define softRight       0x02
#define softLeft        0x10
#define forward         0x12
#define backward        0x21
#define stop            0x00

void setupCLK();
void peripheralEnable();
void gpioEnable();
void interruptEnable();
void encoderInterruptEncountered();
void linearDistanceMM(unsigned int);
void angleRotate(uint16_t);
void forwardMM(unsigned int);
void backwardMM(unsigned int);
void leftDegrees(unsigned int);
void rightDegrees(unsigned int);
void delay_ms(uint64_t delay);
void delay_us(uint64_t delay);
void motion(uint8_t direction);
volatile uint16_t ShaftCountRight=0,ShaftCountLeft=0;

int main(void) {
  setupCLK();
  peripheralEnable();
  gpioEnable();
  interruptEnable();
  while(1){
    forwardMM(100);
    delay_ms(1000);
    rightDegrees(90);
    delay_ms(1000);
  }
}

/*
*********************************************************************************
   * This function is used to setup Clock frequency of the controller
   * It can be changed through codes
   * In this we have set frequency as 40Mhz
   * Frequency is set by SYSDIV which can be found in data sheet for different
frequencies

*********************************************************************************
*/
void setupCLK(){
  SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
}

/*******************************
* Enabling System Peripherals
* PORTF,PORTB and PORTA in this case
******************************/
void peripheralEnable(){
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
}

/*************************************
* Configuring Pin as Input Or Output
* Setting PWM Pins to Always High
* Weak Pull to the Input Pins
************************************/
void gpioEnable(){
  GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_1|GPIO_PIN_2);
  GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_0|GPIO_PIN_1);
  GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE,GPIO_PIN_6|GPIO_PIN_5);
  GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_4|GPIO_PIN_3);
  GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3,255);
  GPIOPinWrite(GPIO_PORTA_BASE,GPIO_PIN_6,255);
  GPIOPinTypeGPIOInput(GPIO_PORTA_BASE, GPIO_PIN_4|GPIO_PIN_3);
  GPIOPadConfigSet(GPIO_PORTA_BASE ,GPIO_PIN_4|GPIO_PIN_5,GPIO_STRENGTH_2MA,GPIO_PIN_TYPE_STD_WPU);
}
/****For Enabling Interrupt on PortA****/
void interruptEnable(){
  GPIOIntDisable(GPIO_PORTA_BASE,GPIO_PIN_4|GPIO_PIN_3);
  GPIOIntClear(GPIO_PORTA_BASE, GPIO_PIN_4|GPIO_PIN_3);
  GPIOIntRegister(GPIO_PORTA_BASE, encoderInterruptEncountered);
  GPIOIntTypeSet(GPIO_PORTA_BASE,GPIO_PIN_4|GPIO_PIN_3,GPIO_FALLING_EDGE);
  GPIOIntEnable(GPIO_PORTA_BASE,GPIO_PIN_4|GPIO_PIN_3);
}
/**** ISR For External Interrupt on PortA************************
* Check on which pin of the PORTA has encountered an interrupt
* There is only one ISR for complete PORT
* No two PORTs can have same ISR
***************************************************************/
void encoderInterruptEncountered(){
  if(GPIOIntStatus(GPIO_PORTA_BASE, false)&GPIO_PIN_4){
    ShaftCountLeft++;
    GPIOIntClear(GPIO_PORTA_BASE, GPIO_PIN_4);
  }
  if(GPIOIntStatus(GPIO_PORTA_BASE, false)&GPIO_PIN_3){
    ShaftCountRight++;
    GPIOIntClear(GPIO_PORTA_BASE, GPIO_PIN_3);
  }
}
/*************************************
* Calculating Delays
************************************/
void delay_ms(uint64_t delay){
  SysCtlDelay(delay*(SysCtlClockGet()/3000));
}
void delay_us(uint64_t delay){
  SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
}
/*****************************************************
* This function is for giving the direction of motion
* Macros have been defined at starting
* Macros for directions are 8 bits
* Out of these 8 bits only 4 are used
* Bit 0 (LSB) corresponds to PB3
* Bit 3       corresponds to PF3
* Bit 4       corresponds to PC4
* Bit 6       corresponds to PF6
***************************************************/
void motion(uint8_t direction){
  GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_0|GPIO_PIN_1,direction);
  GPIOPinWrite(GPIO_PORTA_BASE,GPIO_PIN_5,direction);
  GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_4,direction);
}
/***************************************************
* Function to Rotate to desired Angle
* Resolution can be Change to Get Higher Precision
**************************************************/
void angleRotate(uint16_t Degrees){
  unsigned long int ReqdShaftCountInt = 0;  // division by resolution to get shaft count
  ReqdShaftCountInt = Degrees/ 4.09;;
  ShaftCountRight = 0;
  ShaftCountLeft = 0;
  while (1){
    if((ShaftCountRight>=ReqdShaftCountInt)&&(ShaftCountLeft>=ReqdShaftCountInt))
      break;
  }
  motion(stop);
}
/***************************************************
* Function to Move in a Linear Distance
* Resolution can be Change to Get Higher Precision
**************************************************/
void linearDistanceMM(unsigned int DistanceInMM){
  unsigned long int ReqdShaftCountInt = 0;
  ReqdShaftCountInt =DistanceInMM / 5.338;;
  ShaftCountRight=0;
  ShaftCountLeft=0;
  while(1){
    if((ShaftCountRight > ReqdShaftCountInt)&&(ShaftCountLeft > ReqdShaftCountInt))
      break;
    else if((ShaftCountRight > ReqdShaftCountInt))
      motion(softRight);
    else if((ShaftCountLeft > ReqdShaftCountInt))
    motion(softLeft);
  }
  motion(stop); //Stop robot
}
void forwardMM(unsigned int DistanceInMM){
  motion(forward);
  linearDistanceMM(DistanceInMM);
}
void backwardMM(unsigned int DistanceInMM){
  motion(backward);
  linearDistanceMM(DistanceInMM);
```

```
        }
        void leftDegrees(unsigned int Degrees){
          motion(left); //Turn left
          angleRotate(Degrees);
        }
        void rightDegrees(unsigned int Degrees){
          motion(right); //Turn right
          angleRotate(Degrees);
        }

```

6.6 Timers and its Interrupts

The TM4C123GH6PM General-Purpose Timer Module (GPTM) contains six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks. Each 16/32-bit GPTM block provides two 16-bit timers/counters (referred to as Timer A and Timer B) that can be configured to operate independently as timers or event counters, or concatenated to operate as one 32-bit timer or one 32-bit Real-Time Clock (RTC). Each 32/64-bit Wide GPTM block provides 32-bit timers for Timer A and Timer B that can be concatenated to operate as a 64-bit timer.

Timers are mainly used for

- Velocity Control
- Servo Motor Control
- Event Scheduling
- Velocity Calculation

In this section the event scheduling application of timer is explained. To illustrate this the buzzer is switched On and OFF periodically. The remaining applications are explained in the further sections.

6.6.1 Code

```
        #include <stdint.h>
        #include <stdbool.h>
        #include ”inc/hw_types.h”
        #include ”inc/hw_memmap.h”
        #include ”inc/tm4c123gh6pm.h”
        //This header File is important to Unlock GPIO Pins
        #include ”inc/hw_gpio.h”
        #include ”driverlib/sysctl.h”
        #include ”driverlib/gpio.h”
        //Used for enabling the timer
        #include ”driverlib/timer.h”
        //Used for enabling interrupt
        #include ”driverlib/interrupt.h”
```

```
/**** Useful Macros Definition******/
/******Remove the comments if you are using uC board******
#define buzzerEnable    SYSCTL_PERIPH_GPIOA
#define buzzer          GPIO_PORTA_BASE
#define buzzerPin       GPIO_PIN_2
**********************************************************/

/******Remove the comments if you are using uC board******/
#define buzzerEnable    SYSCTL_PERIPH_GPIOF
#define buzzer          GPIO_PORTF_BASE
#define buzzerPin       GPIO_PIN_4
/**********************************************************/

#define buzzerOn()      GPIOPinWrite(buzzer,buzzerPin,255)
#define buzzerOff()     GPIOPinWrite(buzzer,buzzerPin,0)
/**********************************/
void setupCLK();
void peripheralEnable();
void configIOPin();
void timerEnable();

uint32_t ui32Period;    // used for generating one second delay
volatile int flag = 0;          //used to monitor the state of buzzer

int main(void) {
  setupCLK();
  peripheralEnable();
  configIOPin();
  timerEnable();
  flag = 0;
  while(1){
  }
}
/*************************************************************************************
* This function is used to setup Clock frequency of the controller
* It can be changed through codes
* In this we have set frequency as 40Mhz
* Frequency is set by SYSDIV which can be found in data sheet for different frequencies

**************************************************************************************/
void setupCLK(){
  SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
}
/******************************
* Enabling System Peripherals
* buzzer Port in this case
* buzzerPin for buzzer output
* Enabling Timer 0
******************************/
void peripheralEnable(){
  SysCtlPeripheralEnable(buzzerEnable);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);
  /***** Just in case you are not familiar with macros*****
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
  *************This is enabling PORTF********************/
}
```

```
/***********************************
* Configuring Pin as Input Or Output
***********************************/
void configIOPin(){
  GPIOPinTypeGPIOOutput(buzzer, buzzerPin);
}
/*****************************
* Enabling Timer 0
* Timer is configured to be generate
interrupt every second
* Here sysCtlClockGet() is divided
by the on time of buzzer
*****************************/
void timerEnable(){
  TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
  ui32Period = (SysCtlClockGet() / 1) / 2;
  TimerLoadSet(TIMER0_BASE, TIMER_A, ui32Period -1);
  IntEnable(INT_TIMER0A);
  TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
  IntMasterEnable();
  TimerEnable(TIMER0_BASE, TIMER_A);
}
/***********************************************************
* This function is executed when the timer overflows
* In this example the buzzer is switched on and off alternatively
***********************************************************/
void Timer0IntHandler(void)
{
// Clear the timer interrupt
  TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
  flag = !flag;
  if(flag == 0){
    buzzerOn();
  }
  else{
    buzzerOff();
  }
}

```

6.7 Robot Speed Control

6.7.1 Pulse Width Modulation(PWM)

Pulse width modulation is a process in which duty cycle of constant frequency square wave is modulated to control power delivered to the load i.e. motor.
Duty cycle is the ratio of 'T-ON/ T'. Where 'T-ON' is ON time and 'T' is the time period of the wave. Power delivered to the motor is proportional to the 'T-ON' time of the signal. In case of PWM the motor reacts to the time average of the signal.
PWM is used to control total amount of power delivered to the load without power losses which generally occur in resistive methods of power control.

Above figure shows the PWM waveforms for motor velocity control. In case (A), ON time is 90 percent of time period. This wave has more average value. Hence more power is delivered to themotor. In case (B), the motor will run slower as the ON time is just 10 percent of time period.

The TM4C123GH6PM microcontroller contains two PWM modules, each with four PWM generator blocks and a control block, for a total of 16 PWM outputs. The control block determines the polarity of the PWM signals, and which signals are passed through to the pins. The connections of PWM motor pins are given in the section 6.7. The same code is modified to change the velocity of the motors.

6.7.2 Code for Plug and Play Board:

```
include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/pin_map.h"
//This header File is important to Unlock GPIO Pins
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
//Used for PWM
#include "driverlib/pwm.h"

#define right           0x41
#define left            0x18
#define softRight       0x10
#define softLeft        0x01
#define forward         0x11
#define backward        0x48
#define stop            0x00

void setupCLK();
void peripheralEnable();
void configIOPin();
void delay_ms(uint64_t delay);
void delay_ms(uint64_t delay);
void motion(uint8_t);
void enablePWM();
void Velocity(uint8_t lSpeed,uint8_t rSpeed);

int main(void) {
  setupCLK();
  peripheralEnable();
  configIOPin();
  enablePWM();
  while(1){
    Velocity(150, 150);
    motion(forward);
    delay_ms(2000);
    motion(stop);
    delay_ms(500);
    Velocity(255, 255);
    motion(backward);
    delay_ms(800);
    motion(stop);
    delay_ms(500);
```

```
            Velocity(255, 255);
            motion(right);
            delay_ms(1000);
            motion(stop);
            delay_ms(500);
            Velocity(150, 150);
            motion(left);
            delay_ms(1000);
            motion(stop);
            delay_ms(500);
            Velocity(150, 150);
            motion(backward);
            delay_ms(1000);
          }
        }
        /*
*********************************************************************************
        * This function is used to setup Clock frequency of the controller
        * It can be changed through codes
        * In this we have set frequency as 40Mhz
        * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
        * The PWM module is clocked by the system clock through a divider, and that divider has
        a range of 2 to 64.
        * By setting the divider to 64, it will run the PWM clock at 625 kHz.

*********************************************************************************
*/
        void setupCLK(){
          SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
          SysCtlPWMClockSet(SYSCTL_PWMDIV_64);    //625kHz PWM Clock
        }
        /******************************
        * Enabling System Peripherals
        * PORTF,PORTB and PORTC in this case
        *****************************/
        void peripheralEnable(){
          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
          SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0); // Enabling PWM0
          SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1); // Enabling PWM1
        }
        /***********************************
        * Configuring Pin as Input Or Output
        * And Setting PWM Pin to Always High
        ***********************************/
        void configIOPin(){
          GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_3);
          GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE,GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6);
          GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_3|GPIO_PIN_2);
        }
        /***********************************
        * Calculating Delays
        ***********************************/
        void delay_ms(uint64_t delay){
          SysCtlDelay(delay*(SysCtlClockGet()/3000));
          }
```

```
  void delay_us(uint64_t delay){
  SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
}
/****************************************************
* This function is for giving the direction of motion
* Macros have been defined at starting
* Macros for directions are 8 bits
* Out of these 8 bits only 4 are used
* Bit 0 (LSB) corresponds to PB3
* Bit 3       corresponds to PF3
* Bit 4       corresponds to PC4
* Bit 6       corresponds to PF6
***************************************************/
void motion(uint8_t direction){
  GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_3,direction<<3);
  GPIOPinWrite(GPIO_PORTC_BASE,GPIO_PIN_4|GPIO_PIN_6,direction);
  GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_3,direction);
}
/****************************************************
* This function is for enabling the PWM Modules
* PWM can be enabled on a pin based on the datasheet
***************************************************/
void enablePWM(){
  GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_2);
  GPIOPinConfigure(GPIO_PF2_M1PWM6);
  GPIOPinTypePWM(GPIO_PORTC_BASE, GPIO_PIN_5);
  GPIOPinConfigure(GPIO_PC5_M0PWM7);
  //Count Down Mode
  PWMGenConfigure(PWM0_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
  PWMGenPeriodSet(PWM0_BASE, PWM_GEN_3, 255); //Load Count value
  //Count Down Mode
  PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
  PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, 255); //Load Count value
  PWMGenEnable(PWM0_BASE, PWM_GEN_3);
  PWMGenEnable(PWM1_BASE, PWM_GEN_3);
  PWMOutputState(PWM1_BASE, PWM_OUT_6_BIT, true);
  PWMOutputState(PWM0_BASE, PWM_OUT_7_BIT, true);
}
/***********************************************************
* This function is used to control the speed of the motors
* The speed can changed by the PWMPulseWidthSet() function
* lSpeed is used to control the speed of left motor
* rSpeed is used to control the speed of right motor
**********************************************************/
void Velocity(uint8_t lSpeed,uint8_t rSpeed){
  lSpeed=(lSpeed>255)?255:lSpeed;
  rSpeed=(rSpeed>255)?255:rSpeed;
  PWMPulseWidthSet(PWM1_BASE, PWM_OUT_6, lSpeed);
  PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, rSpeed);
}

```

6.7.3 Code for uC based Board:

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/pin_map.h"
```

```

//This header File is important to Unlock GPIO Pins
#include "inc/hw_gpio.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
//Used for PWM
#include "driverlib/pwm.h"

#define right         0x22
#define left          0x11
#define softRight     0x10
#define softLeft      0x02
#define forward       0x12
#define backward      0x21
#define stop          0x00

void setupCLK();
void peripheralEnable();
void configIOPin();
void delay_ms(uint64_t delay);
void delay_ms(uint64_t delay);
void motion(uint8_t);
void enablePWM();
void Velocity(uint8_t lSpeed,uint8_t rSpeed);

int main(void) {
  setupCLK();
  peripheralEnable();
  configIOPin();
  enablePWM();
  while(1){
    Velocity(150, 150);
    motion(forward);
    delay_ms(2000);
    motion(stop);
    delay_ms(500);
    Velocity(255, 255);
    motion(backward);
    delay_ms(800);
    motion(stop);
    delay_ms(500);
    Velocity(255, 255);
    motion(right);
    delay_ms(1000);
    motion(stop);
    delay_ms(500);
    Velocity(150, 150);
    motion(left);
    delay_ms(1000);
    motion(stop);
    delay_ms(500);
    Velocity(150, 150);
    motion(backward);
    delay_ms(1000);
  }
}
/*
********************************************************************************
* This function is used to setup Clock frequency of the controller
* It can be changed through codes
```

```
        * In this we have set frequency as 40Mhz
        * Frequency is set by SYSDIV which can be found in data sheet for different frequencies
        * * The PWM module is clocked by the system clock through a divider, and that divider has
        a range of 2 to 64.
        * By setting the divider to 64, it will run the PWM clock at 625 kHz.
**********************************************************************************/
        void setupCLK(){
          SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
          SysCtlPWMClockSet(SYSCTL_PWMDIV_64);     //625kHz PWM Clock

        }
        /*******************************
        * Enabling System Peripherals
        * PORTF,PORTB and PORTA in this case
        *******************************/
        void peripheralEnable(){
          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
          SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
          SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1); // Enabling PWM1
        }
        /************************************
        * Configuring Pin as Input Or Output
        * And Setting PWM Pin to Always High
        ************************************/
        void configIOPin(){
          GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE,GPIO_PIN_0|GPIO_PIN_1);
          GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE,GPIO_PIN_5);
          GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,GPIO_PIN_4);
          }
        /************************************
        * Calculating Delays
        ************************************/
        void delay_ms(uint64_t delay){
          SysCtlDelay(delay*(SysCtlClockGet()/3000));
          }
          void delay_us(uint64_t delay){
          SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
          }
        /****************************************************
        * This function is for giving the direction of motion
        * Macros have been defined at starting
        * Macros for directions are 8 bits
        * Out of these 8 bits only 4 are used
        * Bit 0      corresponds to PB0
        * Bit 1      corresponds to PB1
        * Bit 4      corresponds to PF4
        * Bit 5      corresponds to PA5
        ****************************************************/
        void motion(uint8_t direction){
          GPIOPinWrite(GPIO_PORTB_BASE,GPIO_PIN_1|GPIO_PIN_0,direction);
          GPIOPinWrite(GPIO_PORTF_BASE,GPIO_PIN_4,direction);
          GPIOPinWrite(GPIO_PORTA_BASE,GPIO_PIN_5,direction);
        }
        /****************************************************
```

```
* This function is for enabling the PWM Modules
* PWM can be enabled on a pin based on the datasheet
*****************************************************/
void enablePWM(){
  GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_3);
  GPIOPinConfigure(GPIO_PF3_M1PWM7);
  GPIOPinTypePWM(GPIO_PORTA_BASE, GPIO_PIN_6);
  GPIOPinConfigure(GPIO_PA6_M1PWM2);
  //Count Down Mode
  PWMGenConfigure(PWM1_BASE, PWM_GEN_3, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
  PWMGenPeriodSet(PWM1_BASE, PWM_GEN_3, 255); //Load Count value
  //Count Down Mode
  PWMGenConfigure(PWM1_BASE, PWM_GEN_1, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
  PWMGenPeriodSet(PWM1_BASE, PWM_GEN_1, 255); //Load Count value
  PWMGenEnable(PWM1_BASE, PWM_GEN_3); //Enable the generators
  PWMGenEnable(PWM1_BASE, PWM_GEN_1);
  PWMOutputState(PWM1_BASE, PWM_OUT_7_BIT|PWM_OUT_2_BIT, true);
}
/*******************************************************
* This function is used to control the speed of the motors
* The speed can changed by the PWMPulseWidthSet() function
* lSpeed is used to control the speed of left motor
* rSpeed is used to control the speed of right motor
*******************************************************/
void Velocity(uint8_t lSpeed,uint8_t rSpeed){
  lSpeed=(lSpeed>255)?255:lSpeed;
  rSpeed=(rSpeed>255)?255:rSpeed;
  PWMPulseWidthSet(PWM1_BASE, PWM_OUT_7, lSpeed);
  PWMPulseWidthSet(PWM1_BASE, PWM_OUT_2, rSpeed);
}

```

6.8 LCD Interfacing

To interface LCD with the microcontroller in default configuration requires 3 control signals and 8 data lines. This is known as 8 bit interfacing mode which requires total 11 I/O lines. To reduce the number of I/Os required for LCD interfacing we can use 4 bit interfacing mode which requires 3 control signals with 4 data lines. In this mode upper nibble and lower nibble of commands/data set needs to be sent separately. The three control lines are referred to as EN, RS, and RW. The LCD connections are given in section 5.6.

6.8.1 Code for Plug and Play Board:

```
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "inc/hw_gpio.h" //To unlock locked pins for GPIO
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include<math.h>
#include<stdlib.h>
```

```
#ifndef     lcdPORT
#define     lcdPORT       GPIO_PORTD_BASE
#endif
#ifndef     lcdDDR
#define     lcdDDR        GPIO_PORTA_BASE
#endif
#ifndef     lcdPIN
#define     lcdPIN        PINC
#endif
#ifndef     RS
#define     RS            GPIO_PIN_6
#endif
//#ifndef     RW
//#define     RW            GPIO_PIN_1
//#endif
#ifndef     EN
#define     EN            GPIO_PIN_7
#endif
#ifndef     D4
#define     D4            GPIO_PIN_2
#endif
#ifndef     D5
#define     D5            GPIO_PIN_3
#endif
#ifndef     D6
#define     D6            GPIO_PIN_4
#endif
#ifndef     D7
#define     D7            GPIO_PIN_5
#endif
unsigned char cursorPositionCheck=0;

void lcdInit();
void lcdCommand(unsigned char);
void lcdData(unsigned char);
void lcdString(char*);
void lcdGotoxy(unsigned char,unsigned char);
void lcdClear();
void lcdCheck();
void setupCLK();
void peripheralEnable();
void configIOPin();
void _delay_ms(uint64_t delay);
void _delay_us(uint64_t delay);

int main(){
setupCLK();
  peripheralEnable();
  configIOPin();
  lcdInit();
  lcdGotoxy(0,0);
  lcdString(”TIVA C Series”);
  while(1){
  }
}
void setupCLK(){
  SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN)
;
}
void peripheralEnable(){
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
}
void configIOPin(){
  HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
  HWREG(GPIO_PORTD_BASE + GPIO_O_CR) |= (1<<7);
  HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = 0;
  GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE,EN|RS);
  GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE,D4|D5|D6|D7);
}
void lcdInit(){
  lcdCommand(0x28);
  /*************************
  0x30 8bit mode single line*
  0x38 8bit mode double line*
  0x20 4bit mode single line*
  0x28 4bit mode double line*
  **************************/
  lcdCommand(0x06);//entry mode and auto increment mode
  lcdCommand(0x0F);//
  /*******************************
  Display off Cursor off      0x08*
  Display on Cursor on        0x0E*
  Display on Cursor off       0x0C*
  Display on Cursor blinking  0x0F*
  ********************************/
}
void lcdCommand(unsigned char command){
  GPIOPinWrite(lcdPORT,RS|EN,0);
  GPIOPinWrite(lcdDDR,D4|D5|D6|D7,0);
  _delay_us(100);
  GPIOPinWrite(lcdDDR,D4|D5|D6|D7,(command>>2));
  _delay_ms(1);
  GPIOPinWrite(lcdPORT,EN|RS,0x80);
  _delay_us(100);
  GPIOPinWrite(lcdPORT,EN,0);
  _delay_us(100);
  GPIOPinWrite(lcdDDR,D4|D5|D6|D7,(command<<2));;
  _delay_ms(1);
  GPIOPinWrite(lcdPORT,EN|RS,0x80);
  _delay_us(100);
  GPIOPinWrite(lcdPORT,EN,0);
  _delay_us(100);
}
void lcdData(unsigned char data){
  lcdCheck();
  GPIOPinWrite(lcdPORT,RS|EN,0);
  GPIOPinWrite(lcdDDR,D4|D5|D6|D7,0);
  GPIOPinWrite(lcdDDR,D4|D5|D6|D7,(data>>2));
  _delay_us(100);
  GPIOPinWrite(lcdPORT,EN|RS,0xc0);
  _delay_ms(1);
  GPIOPinWrite(lcdPORT,EN,0);
  _delay_us(100);
  GPIOPinWrite(lcdDDR,D4|D5|D6|D7,(data<<2));
  _delay_us(100);
  GPIOPinWrite(lcdPORT,EN|RS,0xc0);
  _delay_us(100);
  GPIOPinWrite(lcdPORT,EN,0);
  cursorPositionCheck=(cursorPositionCheck+1)%32;
```

```
}
void lcdString(char* string){
  unsigned char i=0;
  while(string[i])
  lcdData(string[i++]);
}
void lcdGotoxy(unsigned char x,unsigned char y)
{
  cursorPositionCheck=y*16+x;
  lcdCommand(0x80+x+(64*y));
}
void lcdClear(){
  cursorPositionCheck=0;
  lcdCommand(0x01);
  _delay_ms(3);
}
void lcdCheck(){
  if(cursorPositionCheck==16)
    lcdGotoxy(0,1);
  else if(cursorPositionCheck==0)
    lcdGotoxy(0,0);
}
void _delay_ms(uint64_t delay){
  SysCtlDelay(delay*(SysCtlClockGet()/3000));
}
void _delay_us(uint64_t delay){
  SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
}

```

6.8.2 Code for uC based Board

```
#include <stdint.h>
#include <stdbool.h>
#include ”inc/hw_types.h”
#include ”inc/hw_memmap.h”
#include ”inc/hw_gpio.h” //To unlock locked pins for GPIO
#include ”driverlib/sysctl.h”
#include ”driverlib/gpio.h”
#include<math.h>
#include<stdlib.h>
#ifndef     lcdPORT
#define     lcdPORT     GPIO_PORTF_BASE
#endif
#ifndef     lcdDDR
#define     lcdDDR      GPIO_PORTD_BASE
#endif
#ifndef     lcdPIN
#define     lcdPIN      PINC
#endif
#ifndef     RS
#define     RS          GPIO_PIN_0
#endif
#ifndef     EN
#define     EN          GPIO_PIN_2
#endif
#ifndef     D4
#define     D4          GPIO_PIN_4
#endif
#ifndef     D5
#define     D5          GPIO_PIN_5
```

```
    #endif
    #ifndef     D6
    #define     D6          GPIO_PIN_6
    #endif
    #ifndef     D7
    #define     D7          GPIO_PIN_7
    #endif
    unsigned char cursorPositionCheck=0;

    void lcdInit();
    void lcdCommand(unsigned char);
    void lcdData(unsigned char);
    void lcdString(char*);
    void lcdGotoxy(unsigned char,unsigned char);
    void lcdClear();
    void lcdCheck();
    void setupCLK();
    void peripheralEnable();
    void configIOPin();
    void _delay_ms(uint64_t delay);
    void _delay_us(uint64_t delay);

    int main(){
      setupCLK();
      peripheralEnable();
      configIOPin();
      lcdInit();
      lcdGotoxy(0,0);
      lcdString("TIVA C Series");
      while(1){
      }
    }
    void setupCLK(){
      SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
      }
      void peripheralEnable(){
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    }
    void configIOPin(){
      HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
      HWREG(GPIO_PORTF_BASE + GPIO_O_CR) |= 0x01;
      HWREG(GPIO_PORTF_BASE + GPIO_O_LOCK) = 0;
      GPIOPinTypeGPIOOutput(lcdPORT,EN|RS);
      GPIOPinTypeGPIOOutput(lcdDDR,D4|D5|D6|D7);
    }
    void lcdInit(){
      lcdCommand(0x28);
      /************************
      0x30 8bit mode single line*
      0x38 8bit mode double line*
      0x20 4bit mode single line*
      0x28 4bit mode double line*
      *************************/
      lcdCommand(0x06);//entry mode and auto increment mode
      lcdCommand(0x0F);//
      /******************************
      Display off Cursor off      0x08*
      Display on Cursor on        0x0E*
```

```
  Display on Cursor off        0x0C*
  Display on Cursor blinking   0x0F*
  ********************************/
}
void lcdCommand(unsigned char command){
  GPIOPinWrite(lcdPORT,RS|EN,0);
  GPIOPinWrite(lcdDDR,D4|D5|D6|D7,0);
  _delay_us(100);
  GPIOPinWrite(lcdDDR,D4|D5|D6|D7,command);
  _delay_us(100);
  GPIOPinWrite(lcdPORT,EN|RS,0x04);
  _delay_ms(1);
  GPIOPinWrite(lcdPORT,EN,0);
  _delay_us(100);
  GPIOPinWrite(lcdDDR,D4|D5|D6|D7,(command<<4));
  _delay_us(100);
  GPIOPinWrite(lcdPORT,EN|RS,0x04);
  _delay_ms(1);
  GPIOPinWrite(lcdPORT,EN,0);
  _delay_us(100);
}
void lcdData(unsigned char data){
  lcdCheck();
  GPIOPinWrite(lcdPORT,RS|EN,0);
  GPIOPinWrite(lcdDDR,D4|D5|D6|D7,0);
  GPIOPinWrite(lcdDDR,D4|D5|D6|D7,data);
  _delay_us(100);
  GPIOPinWrite(lcdPORT,EN|RS,0x05);
  _delay_ms(1);
  GPIOPinWrite(lcdPORT,EN,0);
  _delay_us(100);
  GPIOPinWrite(lcdDDR,D4|D5|D6|D7,(data<<4));
  _delay_us(100);
  GPIOPinWrite(lcdPORT,EN|RS,0x05);
  _delay_ms(1);
  GPIOPinWrite(lcdPORT,EN,0);
  cursorPositionCheck=(cursorPositionCheck+1)%32;
}
void lcdString(char* string){
  unsigned char i=0;
  while(string[i])
    lcdData(string[i++]);
}
void lcdGotoxy(unsigned char x,unsigned char y)
{
  cursorPositionCheck=y*16+x;
  lcdCommand(0x80+x+(64*y));
}
void lcdClear(){
  cursorPositionCheck=0;
  lcdCommand(0x01);
  _delay_ms(3);
}
void lcdCheck(){
  if(cursorPositionCheck==16)
    lcdGotoxy(0,1);
  else if(cursorPositionCheck==0)
    lcdGotoxy(0,0);
}
void _delay_ms(uint64_t delay){
  SysCtlDelay(delay*(SysCtlClockGet()/3000));
}
void _delay_us(uint64_t delay){
  SysCtlDelay(delay*(SysCtlClockGet()/3000000UL));
}

```

6.9 Analog To Digital Converter

Fire Bird V has three white line sensors, one Sharp IR range sensor with four add-on sockets for additional Sharp IR range sensors, eight Analog IR proximity sensors. All these sensors give analog output. We need to use ADC (Analog to Digital Converter) to convert these analog values in to digital values.

The TM4C123GH6PM ADC module features 12-bit conversion resolution and supports 12 input channels, plus an internal temperature sensor. Each ADC module contains four programmable sequencers allowing the sampling of multiple analog input sources without controller intervention. Each sample sequencer provides flexible programming with fully configurable input source, trigger events, interrupt generation, and sequencer priority.

Due to limiited number of sensors in TM4C123GH6PM, an external ADC(ADC128d818) is added to the daughter Board. Details about interfacing this module is given in the next section.

The Connections of internal ADC is as shown below

6.9.1 Code for Plug and Play Board:

```
  #include <stdint.h>
  #include <stdbool.h>
  #include "stdlib.h"
  #include "inc/hw_ints.h"
  #include "inc/hw_memmap.h"
  #include "inc/hw_uart.h"
  #include "inc/hw_gpio.h"
  #include "inc/hw_pwm.h"
  #include "inc/hw_types.h"
  #include "driverlib/adc.h"
  #include "driverlib/timer.h"
  #include "driverlib/gpio.h"
  #include "driverlib/interrupt.h"
  #include "driverlib/pin_map.h"
  #include "driverlib/rom.h"
  #include "driverlib/rom_map.h"
  #include "driverlib/sysctl.h"
  #include "driverlib/uart.h"
  #include "driverlib/udma.h"
  #include "driverlib/pwm.h"
  #include "driverlib/ssi.h"
  #include "driverlib/systick.h"
  #include "driverlib/adc.h"
  #include "utils/uartstdio.h"
  #include "utils/uartstdio.c"
```

```
    #include <string.h>

    void configCLK();
    void peripheralEnable();
    void uartEnable();

    void ADC0Enable();
    unsigned int readADC();
    void tranString(char * data,char delimeter);
    void uartInteger(long long int integer,char delimeter);
    void converter(unsigned int);
    void _delay_ms(uint64_t delay);
    uint32_t senval;

    int main(){
      configCLK();
      peripheralEnable();
      ADC0Enable();
      uartEnable();
      while(1){
        senval = readADC();
        converter(senval);
        _delay_ms(1000);
      }
    }
    /*
********************************************************************************
    * This function is used to setup Clock frequency of the controller
    * It can be changed through codes
    * In this we have set frequency as 40Mhz
    * Frequency is set by SYSDIV which can be found in data sheet for different
frequencies

********************************************************************************
*/
    void configCLK(){
      SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ)
;
    }
    /*****************************
    * Enabling System Peripherals
    * PortB and PortD in this case
    ****************************/
    void peripheralEnable(){
      SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);//Enabling TIMER0
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
      SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
      ADCHardwareOversampleConfigure(ADC0_BASE, 64);
    }
    /**************************************
    * This function is used to enable UART1
    * The baudrate is set at 9600
    *************************************/
    void uartEnable(){
      GPIOPinConfigure(GPIO_PB0_U1RX);     //Configure Pin B0 as RX of U0
      GPIOPinConfigure(GPIO_PB1_U1TX);     //Configure Pin B1 as TX of U0
      GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
      UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 9600,(UART_CONFIG_WLEN_8 |
UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
}
/*****************************************************
* This function is used to enable ADC0
* 4 step sequencer is used
* Change the channel number to use any of the other ADCs
*****************************************************/
void ADC0Enable(){
  ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
  ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_CH4);
  ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_CH4);
  ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_CH4);
  ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_CH4|ADC_CTL_IE|ADC_CTL_END);
  ADCSequenceEnable(ADC0_BASE, 1);
  GPIOPinTypeADC(GPIO_PORTD_BASE, GPIO_PIN_3);
}
/*********************************************************
* This function is used to read the value from ADC
* Average of 4 values is returned to the calling function
*********************************************************/
unsigned int readADC(){
  unsigned int Avg;
  uint32_t ADC0Value[4];
  ADCIntClear(ADC0_BASE, 1);
  ADCProcessorTrigger(ADC0_BASE, 1);
  while(!ADCIntStatus(ADC0_BASE, 1, false));
  ADCSequenceDataGet(ADC0_BASE, 1, ADC0Value);
  Avg = (ADC0Value[0] + ADC0Value[1] + ADC0Value[2] + ADC0Value[3] + 2)/4;
  return(Avg);
}
/*********************************************************
* This function is used to send the ADC values through UART
* Here the value is sent in reverse order
*********************************************************/
void converter(uint32_t q)
{
  unsigned int p;
  p=q;
  do
  {
    p = (q % 10);
    UARTCharPut(UART1_BASE,48+(int)p);
    SysCtlDelay(400000);
    q = q / 10;
  }while(q != 0);
  UARTCharPut(UART1_BASE,' ');
}
/************************************
* Calculating Delays
************************************/
void _delay_ms(uint64_t delay){
  SysCtlDelay(delay*(SysCtlClockGet()/3000));
}

```

6.9.2 Code for uC based Board

```
#include <stdint.h>
#include <stdbool.h>
#include "stdlib.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
```

```
#include "inc/hw_uart.h"
#include "inc/hw_gpio.h"
#include "inc/hw_pwm.h"
#include "inc/hw_types.h"
#include "driverlib/adc.h"
#include "driverlib/timer.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/udma.h"
#include "driverlib/pwm.h"
#include "driverlib/ssi.h"
#include "driverlib/systick.h"
#include "driverlib/adc.h"
#include "utils/uartstdio.h"
#include "utils/uartstdio.c"
#include <string.h>
#include <math.h>

void configCLK();
void peripheralEnable();
void uartEnable();
unsigned int Sharp_GP2D12_estimation(uint16_t adc_reading);
void ADC0Enable();
unsigned int readADC();
void tranString(char * data,char delimeter);
void uartInteger(long long int integer,char delimeter);
void _delay_ms(uint64_t delay);
void itoa(long long a,char *arr);

int main(){
  configCLK();
  peripheralEnable();
  ADC0Enable();
  uartEnable();

  while(1){
    uartInteger(Sharp_GP2D12_estimation(readADC()),' ');
    _delay_ms(1000);
  }
}
/**************************************************************************************************
* This function is used to setup Clock frequency of the controller
* It can be changed through codes
* In this we have set frequency as 40Mhz
* Frequency is set by SYSDIV which can be found in data sheet for different frequencies

**************************************************************************************************/
void configCLK(){
  SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
}
/******************************
```

```
	* Enabling System Peripherals
	* PortB and PortD in this case
	*****************************/
	void peripheralEnable(){
	  SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
	  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
	  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
	  SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
	  ADCHardwareOversampleConfigure(ADC0_BASE, 64);
	}
	/****************************************
	* This function is used to enable UART1
	* The baudrate is set at 9600
	****************************************/
	void uartEnable(){
	  GPIOPinConfigure(GPIO_PC4_U1RX);    //Configure Pin B0 as RX of U0
	  GPIOPinConfigure(GPIO_PC5_U1TX);    //Configure Pin B1 as TX of U0
	  GPIOPinTypeUART(GPIO_PORTC_BASE, GPIO_PIN_5 | GPIO_PIN_4);
	  UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 9600,(UART_CONFIG_WLEN_8 |
    UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
	}
	/*****************************************************
	* This function is used to enable ADC0
	* 4 step sequencer is used
	* Change the channel number to use any of the other ADCs
	*****************************************************/
	void ADC0Enable(){
	  ADCSequenceConfigure(ADC0_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
	  ADCSequenceStepConfigure(ADC0_BASE, 1, 0, ADC_CTL_CH1);
	  ADCSequenceStepConfigure(ADC0_BASE, 1, 1, ADC_CTL_CH1);
	  ADCSequenceStepConfigure(ADC0_BASE, 1, 2, ADC_CTL_CH1);
	  ADCSequenceStepConfigure(ADC0_BASE,1,3,ADC_CTL_CH1|ADC_CTL_IE|ADC_CTL_END);
	  ADCSequenceEnable(ADC0_BASE, 1);
	  GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_2);
	}
	/**********************************************************
	* This function is used to read the value from ADC
	* Average of 4 values is returned to the calling function
	**********************************************************/
	unsigned int readADC(){
	  unsigned int Avg;
	  uint32_t ADC0Value[4];
	  ADCIntClear(ADC0_BASE, 1);
	  ADCProcessorTrigger(ADC0_BASE, 1);
	  while(!ADCIntStatus(ADC0_BASE, 1, false));
	  ADCSequenceDataGet(ADC0_BASE, 1, ADC0Value);
	  Avg = (ADC0Value[0] + ADC0Value[1] + ADC0Value[2] + ADC0Value[3] + 2)/4;
	  return(Avg);
	}
	void itoa(long long a,char *arr){
	  int i=0,j=0;
	  long long tmp=a;
	  if(a<0){
	    arr[i++]='-';
	    tmp*=-1;
	    j=1;
	  }
	  for(;tmp>0;i++){
	    arr[i]=(tmp%10)+'0';
	    tmp/=10;
```

```
      }
      arr[i--]='\0';
      for(;j<i;j++,i--){
        tmp=arr[i];
        arr[i]=arr[j];
        arr[j]=tmp;
      }
    }
    /************************************
    * Calculating Delays
    ************************************/
    void _delay_ms(uint64_t delay){
      SysCtlDelay(delay*(SysCtlClockGet()/3000));
    }
    void uartInteger(long long int integer,char delimeter){
      char ch[20];
      itoa(integer,ch);
      tranString(ch,delimeter);
    }
    void tranString(char *data,char delimeter){
      int k=0;
      while(data[k]){
        UARTCharPut(UART1_BASE,data[k++]);
      }
      UARTCharPut(UART1_BASE,delimeter);
      }
    unsigned int Sharp_GP2D12_estimation(uint16_t adc_reading){
      float distance;
      unsigned int distanceInt;
      distance = (int)(10.00*(2799.6*(1.00/(pow(adc_reading,1.1546)))));
      distanceInt = (int)distance;
      if(distanceInt>800){
        distanceInt=800;
      }
      return distanceInt;
    }

```

6.10 Serial Communication

The Fire Bird V can communicate with other robots / devices serially using either wired link or wireless module. Serial communication is done in asynchronous mode. In the asynchronous mode, the common clock signal is not required at both the transmitter and receiver for data synchronization. As an example of serial communication code for interfacing Zigbee module is given below.

6.10.1 Connections

6.10.2 Code for Plug and Play Board:

```
      #include <stdlib.h>
      #include <stdint.h>
```

```
#include <stdbool.h>
#include<math.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/debug.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "inc/tm4c123gh6pm.h"

void configCLK();
void peripheralEnable();
void uartEnable();
void uartInterruptEnable();
void UARTIntHandler(void);
void tranString(char *,char);
void uartInteger(int64_t number);

int main(){
  configCLK();
  peripheralEnable();
  uartEnable();
  uartInterruptEnable();
  while(1){
    tranString("Hello",' ');
  }
}
void configCLK(){
  SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
}
void peripheralEnable(){
  SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);//Enablinig TIMER0
}
void uartEnable(){
  GPIOPinConfigure(GPIO_PB0_U1RX);//Configure Pin PB0 as RX of U0
  GPIOPinConfigure(GPIO_PB1_U1TX);//Configure Pin PB1 as TX of U0
  GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
  UARTConfigSetExpClk(UART1_BASE, SysCtlClockGet(), 9600,
  (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
}
void tranString(char * data,char delimeter){
  int k=0;
  while(data[k]){
    UARTCharPut(UART1_BASE,data[k++]);
  }
  UARTCharPut(UART1_BASE,delimeter);
}
void uartInterruptEnable(){
  IntMasterEnable();//Enable processor interrupt
  IntEnable(INT_UART1);//Enable interrupt on UART0
  UARTIntEnable(UART1_BASE, UART_INT_RX | UART_INT_RT);//Enable RX interrupt ant rx Timeout interrupt
}
void UARTIntHandler(void){
  uint32_t ui32Status;
  ui32Status = UARTIntStatus(UART1_BASE, true); //get interrupt status
```

```
      UARTIntClear(UART1_BASE, ui32Status); //clear the asserted interrupts
      while(UARTCharsAvail(UART1_BASE)){ //loop while there are chars
        UARTCharPut(UART1_BASE, UARTCharGet(UART1_BASE));
      }
    }

```

6.10.3 Code for Plug and Play Board:

```
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include<math.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/debug.h"
#include "driverlib/interrupt.h"
#include "driverlib/gpio.h"
#include "inc/tm4c123gh6pm.h"

void configCLK();
void peripheralEnable();
void uartEnable();
void uartInterruptEnable();
void UARTIntHandler(void);
void tranString(char *,char);
void uartInteger(int64_t number);

int main(){
  configCLK();
  peripheralEnable();
  uartEnable();
  uartInterruptEnable();
  while(1){
  }
}
void configCLK(){
  SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_OSC_MAIN|SYSCTL_XTAL_16MHZ);
}
void peripheralEnable(){
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART3);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);//Enablinig TIMER0
}
void uartEnable(){
  GPIOPinConfigure(GPIO_PC6_U3RX);//Configure Pin PC6 as RX of U0
  GPIOPinConfigure(GPIO_PC7_U3TX);//Configure Pin PC7 as TX of U0
  GPIOPinTypeUART(GPIO_PORTC_BASE, GPIO_PIN_6 | GPIO_PIN_7);
  UARTConfigSetExpClk(UART3_BASE, SysCtlClockGet(), 9600,
  (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));
}
void tranString(char * data,char delimeter){
  int k=0;
  while(data[k]){
    UARTCharPut(UART3_BASE,data[k++]);
  }
  UARTCharPut(UART3_BASE,delimeter);
}
```

```
    void uartInterruptEnable(){
      IntMasterEnable();//Enable processor interrupt
      IntEnable(INT_UART3);//Enable interrupt on UART0
      UARTIntEnable(UART3_BASE, UART_INT_RX | UART_INT_RT);//Enable RX interrupt ant rx Timeout interrupt
    }
    void UARTIntHandler(void){
      uint32_t ui32Status;
      ui32Status = UARTIntStatus(UART3_BASE, true); //get interrupt status
      UARTIntClear(UART3_BASE, ui32Status); //clear the asserted interrupts
      while(UARTCharsAvail(UART3_BASE)){ //loop while there are chars
        UARTCharPut(UART3_BASE, UARTCharGet(UART3_BASE));
      }
    }


```

6.11 I2C Communication

6.11.1 Introduction

I2C is a serial protocol for two-wire interface to connect low-speed devices like microcontrollers, EEPROMs, A/D and D/A converters, I/O interfaces and other similar peripherals in embedded systems. I2C combines the best features of SPI and UARTs. Like the Serial Peripheral Interface (SPI), it is only intended for short distance communications within a single device. Like Asynchronous Serial Interfaces (such as RS-232 or UARTs), it only requires two signal wires to exchange information.

6.11.2 Features

- 2 bidirectional lines SDA (serial data) and SCL (serial clock)
- Independent Master, Slave, and Monitor functions
- Supports both Multi-master and Multi-master with Slave functions
- One slave address can be selectively qualified with a bit mask or an address range in order to respond to multiple I2C bus addresses
- advanced features such as automatic multi master arbitration management

6.11.3 Working

In normal state both lines SDA and SCL are at high state this indicates the bus is free, any device can use this bus. Master will now initiate the transfer by triggering a start condition with slave address to which it want to communicate. The corresponding slave device matches its address with master sent address, if it matches then both starts

communicating.
If logic low is sent for bit 0 then the master writes data to the slave device, otherwise next byte wise of the remaining data will be read from the slave device. After all the communication between the master and slave is over master will generate a stop condition indicating that communication is over and another slave can use the bus.

As an example of I2C communication code for interfacing port expander IC is given below. In this case MCP23017 I2C based IC is used.The data sheet of the IC can be downloaded here MCP23017

6.11.4 Connections

6.11.5 Code for Plug and Play Board:

```
#include <stdarg.h>
#include <stdbool.h>
#include <stdint.h>
#include ”inc/hw_i2c.h”
#include ”inc/hw_memmap.h”
#include ”inc/hw_types.h”
#include ”inc/hw_gpio.h”
#include ”driverlib/i2c.h”
#include ”driverlib/sysctl.h”
#include ”driverlib/gpio.h”
#include ”driverlib/pin_map.h”
#include ”inc/tm4c123gh6pm.h”
#include ”driverlib/interrupt.h”
void setupCLK();
void peripheralEnable();
void gpioEnable();
void InitI2C1(void);
void I2CSendString(uint32_t slave_addr, char array[]);
void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...);
uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg);
void portExpanderIO(unsigned char port,unsigned char pin);
void portExpanderSetOutput(unsigned char,unsigned char);
unsigned char portExpanderReadInput(unsigned char);
void portExpanderInterruptEnableAnyChange(unsigned char,unsigned char);
void portExpanderpullup(unsigned char,unsigned char);
void portExpanderInterruptHandler();
int main(void) {
setupCLK();
peripheralEnable();
gpioEnable();
InitI2C1();
portExpanderIO(0x00,0xff);
portExpanderIO(0x01,0x00);
portExpanderSetOutput(0x01,0x00);
portExpanderpullup(0x00,0x0f);
portExpanderInterruptEnableAnyChange(0x00,0xff);
while(1){
}
```

```
	}
	void setupCLK(){
	SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN);
	}
	void peripheralEnable(){
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
	}
	void gpioEnable(){
	GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_7);;
	GPIOPinTypeGPIOInput(GPIO_PORTC_BASE, GPIO_PIN_7);
	GPIOPadConfigSet(GPIO_PORTC_BASE ,GPIO_PIN_7,GPIO_STRENGTH_2MA, GPIO_PIN_TYPE_STD_WPU);
	}
	void InitI2C1(void){
	SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C1);
	SysCtlPeripheralReset(SYSCTL_PERIPH_I2C1);
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
	GPIOPinConfigure(GPIO_PA6_I2C1SCL);
	GPIOPinConfigure(GPIO_PA7_I2C1SDA);
	GPIOPinTypeI2CSCL(GPIO_PORTA_BASE, GPIO_PIN_6);
	GPIOPinTypeI2C(GPIO_PORTA_BASE, GPIO_PIN_7);
	// Enable and initialize the I2C1 master module.  Use the system clock for
	// the I2C1 module.  The last parameter sets the I2C data transfer rate.
	// If false the data rate is set to 100kbps and if true the data rate will
	// be set to 400kbps.
	I2CMasterInitExpClk(I2C1_BASE, SysCtlClockGet(), false);
	//clear I2C FIFOs
	HWREG(I2C1_BASE + I2C_O_FIFOCTL) = 80008000;
	I2CSend(0x20,2,0x0A,1<<6);
	}
	void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...)
	{
	// Tell the master module what address it will place on the bus when
	// communicating with the slave.
	I2CMasterSlaveAddrSet(I2C1_BASE, slave_addr, false);

	//stores list of variable number of arguments
	va_list vargs;

	//specifies the va_list to ”open” and the last fixed argument
	//so vargs knows where to start looking
	va_start(vargs, num_of_args);

	//put data to be sent into FIFO
	I2CMasterDataPut(I2C1_BASE, va_arg(vargs, uint32_t));

	//if there is only one argument, we only need to use the
	//single send I2C function
	if(num_of_args == 1)
	{
	//Initiate send of data from the MCU
	I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_SEND);

	// Wait until MCU is done transferring.
	while(I2CMasterBusy(I2C1_BASE));

	//”close” variable argument list
	va_end(vargs);
	}
```

```
        //otherwise, we start transmission of multiple bytes on the
        //I2C bus
        else
        {
        //Initiate send of data from the MCU
        I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_START);

        // Wait until MCU is done transferring.
        while(I2CMasterBusy(I2C1_BASE));

        //send num_of_args-2 pieces of data, using the
        //BURST_SEND_CONT command of the I2C module
        uint8_t i ;
        for(i = 1; i < (num_of_args - 1); i++)
        {
        //put next piece of data into I2C FIFO
        I2CMasterDataPut(I2C1_BASE, va_arg(vargs, uint32_t));
        //send next data that was just placed into FIFO
        I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);

        // Wait until MCU is done transferring.
        while(I2CMasterBusy(I2C1_BASE));
        }

        //put last piece of data into I2C FIFO
        I2CMasterDataPut(I2C1_BASE, va_arg(vargs, uint32_t));
        //send next data that was just placed into FIFO
        I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
        // Wait until MCU is done transferring.
        while(I2CMasterBusy(I2C1_BASE));

        //"close" variable args list
        va_end(vargs);
        }
        }
        //sends an array of data via I2C to the specified slave
        void I2CSendString(uint32_t slave_addr, char array[])
        {
        // Tell the master module what address it will place on the bus when
        // communicating with the slave.
        I2CMasterSlaveAddrSet(I2C1_BASE, slave_addr, false);

        //put data to be sent into FIFO
        I2CMasterDataPut(I2C1_BASE, array[0]);

        //if there is only one argument, we only need to use the
        //single send I2C function
        if(array[1] == '\0')
        {
        //Initiate send of data from the MCU
        I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_SEND);

        // Wait until MCU is done transferring.
        while(I2CMasterBusy(I2C1_BASE));
        }

        //otherwise, we start transmission of multiple bytes on the
        //I2C bus
        else
```

```
{
//Initiate send of data from the MCU
I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_START);

// Wait until MCU is done transferring.
while(I2CMasterBusy(I2C1_BASE));

//initialize index into array
uint8_t i = 1;

//send num_of_args-2 pieces of data, using the
//BURST_SEND_CONT command of the I2C module
while(array[i + 1] != '\0')
{
//put next piece of data into I2C FIFO
I2CMasterDataPut(I2C1_BASE, array[i++]);

//send next data that was just placed into FIFO
I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);

// Wait until MCU is done transferring.
while(I2CMasterBusy(I2C1_BASE));
}

//put last piece of data into I2C FIFO
I2CMasterDataPut(I2C1_BASE, array[i]);

//send next data that was just placed into FIFO
I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);

// Wait until MCU is done transferring.
while(I2CMasterBusy(I2C1_BASE));
}
}
//read specified register on slave device
uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg)
{
//specify that we are writing (a register address) to the
//slave device
I2CMasterSlaveAddrSet(I2C1_BASE, slave_addr, false);

//specify register to be read
I2CMasterDataPut(I2C1_BASE, reg);

//send control byte and register address byte to slave device
I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_START);

//wait for MCU to finish transaction
while(I2CMasterBusy(I2C1_BASE));

//specify that we are going to read from slave device
I2CMasterSlaveAddrSet(I2C1_BASE, slave_addr, true);

//send control byte and read from the register we
//specified
I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);

//wait for MCU to finish transaction
while(I2CMasterBusy(I2C1_BASE));

```

```
        //return data pulled from the specified register
        return I2CMasterDataGet(I2C1_BASE);
        }
        void portExpanderIO(unsigned char port,unsigned char pin){
        I2CSend(0x20,2,port,pin);
        }
        void portExpanderSetOutput(unsigned char port,unsigned char pin){
        I2CSend(0x20,2,port+(0x12),pin);
        }
        unsigned char portExpanderReadInput(unsigned char port){
        return(I2CReceive(0x20,(port+12)));
        }
        void portExpanderInterruptEnableAnyChange(unsigned char port,unsigned char pin){
        portExpanderIO(port,pin);
        I2CSend(0x20,2,(0x04)+port,pin);
        I2CSend(0x20,2,(0x08)+port,pin);
        GPIOIntDisable(GPIO_PORTC_BASE,GPIO_PIN_7);        // Disable interrupt for PF4 (in case it was enabled)
        GPIOIntClear(GPIO_PORTC_BASE,GPIO_PIN_7);      // Clear pending interrupts for PF4
        GPIOIntRegister(GPIO_PORTC_BASE,portExpanderInterruptHandler);     // Register our handler function for port F
        GPIOIntTypeSet(GPIO_PORTC_BASE,GPIO_PIN_7,GPIO_FALLING_EDGE);            // Configure PF4 for falling edge trigger
        GPIOIntEnable(GPIO_PORTC_BASE,GPIO_PIN_7);
        }
        void portExpanderInterruptHandler(){
        if(GPIOIntStatus(GPIO_PORTC_BASE, false)&GPIO_PIN_7){
        if(I2CReceive(0x20,0x0e)&0x01==0x01){
        portExpanderSetOutput(0x01,0xff);
        }
        I2CReceive(0x20,0x10);
        I2CReceive(0x20,0x11);
        GPIOIntClear(GPIO_PORTC_BASE,GPIO_PIN_7);
        }
        }
        void portExpanderpullup(unsigned char port,unsigned char pin){
        I2CSend(0x20,2,(0x0C)+port,pin);
        }

```

6.11.6 Code for uC based Board:

```
        #include <stdarg.h>
        #include <stdbool.h>
        #include <stdint.h>
        #include "inc/hw_i2c.h"
        #include "inc/hw_memmap.h"
        #include "inc/hw_types.h"
        #include "inc/hw_gpio.h"
        #include "driverlib/i2c.h"
        #include "driverlib/sysctl.h"
        #include "driverlib/gpio.h"
        #include "driverlib/pin_map.h"
        #include "inc/tm4c123gh6pm.h"
        #include "driverlib/interrupt.h"
        void setupCLK();
        void peripheralEnable();
        void gpioEnable();
        void InitI2C0(void);
        void I2CSendString(uint32_t slave_addr, char array[]);
        void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...);
        uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg);
        void portExpanderIO(unsigned char port,unsigned char pin);
        void portExpanderSetOutput(unsigned char,unsigned char);
        unsigned char portExpanderReadInput(unsigned char);
        void portExpanderInterruptEnableAnyChange(unsigned char,unsigned char);
        void portExpanderpullup(unsigned char,unsigned char);
        void portExpanderInterruptHandler();
        int main(void) {
        setupCLK();
        peripheralEnable();
        gpioEnable();
        InitI2C0();
        portExpanderIO(0x00,0xff);
        portExpanderIO(0x01,0x00);
        portExpanderSetOutput(0x01,0x00);
        portExpanderpullup(0x00,0x0f);
        portExpanderInterruptEnableAnyChange(0x00,0xff);
        while(1){
        }
        }
        void setupCLK(){
        SysCtlClockSet(SYSCTL_SYSDIV_5|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|SYSCTL_OSC_MAIN)
;
        }
        void peripheralEnable(){
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
        }
        void gpioEnable(){
        GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, GPIO_PIN_7);;
        GPIOPinTypeGPIOInput(GPIO_PORTC_BASE, GPIO_PIN_7);
        GPIOPadConfigSet(GPIO_PORTC_BASE ,GPIO_PIN_7,GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPU);
        }
        void InitI2C0(void){
        SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
        SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
        GPIOPinConfigure(GPIO_PB2_I2C0SCL);
        GPIOPinConfigure(GPIO_PB3_I2C0SDA);
        GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
        GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);
        // Enable and initialize the I2C0 master module.  Use the system clock for
        // the I2C0 module.  The last parameter sets the I2C data transfer rate.
        // If false the data rate is set to 100kbps and if true the data rate will
        // be set to 400kbps.
        I2CMasterInitExpClk(I2C0_BASE, SysCtlClockGet(), false);
        //clear I2C FIFOs
       HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 80008000;
        I2CSend(0x20,2,0x0A,1<<6);
        }
        void I2CSend(uint8_t slave_addr, uint8_t num_of_args, ...)
        {
        // Tell the master module what address it will place on the bus when
        // communicating with the slave.
        I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);

        //stores list of variable number of arguments
        va_list vargs;

```

```
//specifies the va_list to ”open” and the last fixed argument
//so vargs knows where to start looking
va_start(vargs, num_of_args);

//put data to be sent into FIFO
I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));

//if there is only one argument, we only need to use the
//single send I2C function
if(num_of_args == 1)
{
//Initiate send of data from the MCU
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);

// Wait until MCU is done transferring.
while(I2CMasterBusy(I2C0_BASE));

//”close” variable argument list
va_end(vargs);
}

//otherwise, we start transmission of multiple bytes on the
//I2C bus
else
{
//Initiate send of data from the MCU
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);

// Wait until MCU is done transferring.
while(I2CMasterBusy(I2C0_BASE));

//send num_of_args-2 pieces of data, using the
//BURST_SEND_CONT command of the I2C module
uint8_t i ;
for(i = 1; i < (num_of_args - 1); i++)
{
//put next piece of data into I2C FIFO
I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
//send next data that was just placed into FIFO
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);

// Wait until MCU is done transferring.
while(I2CMasterBusy(I2C0_BASE));
}

//put last piece of data into I2C FIFO
I2CMasterDataPut(I2C0_BASE, va_arg(vargs, uint32_t));
//send next data that was just placed into FIFO
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
// Wait until MCU is done transferring.
while(I2CMasterBusy(I2C0_BASE));

//”close” variable args list
va_end(vargs);
}
}
//sends an array of data via I2C to the specified slave
void I2CSendString(uint32_t slave_addr, char array[])
{
// Tell the master module what address it will place on the bus when
// communicating with the slave.
I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);

//put data to be sent into FIFO
I2CMasterDataPut(I2C0_BASE, array[0]);

//if there is only one argument, we only need to use the
//single send I2C function
if(array[1] == '\0')
{
//Initiate send of data from the MCU
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);

// Wait until MCU is done transferring.
while(I2CMasterBusy(I2C0_BASE));
}

//otherwise, we start transmission of multiple bytes on the
//I2C bus
else
{
//Initiate send of data from the MCU
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);

// Wait until MCU is done transferring.
while(I2CMasterBusy(I2C0_BASE));

//initialize index into array
uint8_t i = 1;

//send num_of_args-2 pieces of data, using the
//BURST_SEND_CONT command of the I2C module
while(array[i + 1] != '\0')
{
//put next piece of data into I2C FIFO
I2CMasterDataPut(I2C0_BASE, array[i++]);

//send next data that was just placed into FIFO
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_CONT);

// Wait until MCU is done transferring.
while(I2CMasterBusy(I2C0_BASE));
}

//put last piece of data into I2C FIFO
I2CMasterDataPut(I2C0_BASE, array[i]);

//send next data that was just placed into FIFO
I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);

// Wait until MCU is done transferring.
while(I2CMasterBusy(I2C0_BASE));
}
}
//read specified register on slave device
uint32_t I2CReceive(uint32_t slave_addr, uint8_t reg)
{
//specify that we are writing (a register address) to the
//slave device
I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, false);
```

```

        //specify register to be read
        I2CMasterDataPut(I2C0_BASE, reg);

        //send control byte and register address byte to slave device
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);

        //wait for MCU to finish transaction
        while(I2CMasterBusy(I2C0_BASE));

        //specify that we are going to read from slave device
        I2CMasterSlaveAddrSet(I2C0_BASE, slave_addr, true);

        //send control byte and read from the register we
        //specified
        I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);

        //wait for MCU to finish transaction
        while(I2CMasterBusy(I2C0_BASE));

        //return data pulled from the specified register
        return I2CMasterDataGet(I2C0_BASE);
        }
        void portExpanderIO(unsigned char port,unsigned char pin){
        I2CSend(0x20,2,port,pin);
        }
        void portExpanderSetOutput(unsigned char port,unsigned char pin){
        I2CSend(0x20,2,port+(0x12),pin);
        }
        unsigned char portExpanderReadInput(unsigned char port){
        return(I2CReceive(0x20,(port+12)));
        }
        void portExpanderInterruptEnableAnyChange(unsigned char port,unsigned char pin){
        portExpanderIO(port,pin);
        I2CSend(0x20,2,(0x04)+port,pin);
        I2CSend(0x20,2,(0x08)+port,pin);
        GPIOIntDisable(GPIO_PORTC_BASE,GPIO_PIN_7);        // Disable interrupt for PF4 (in case it was enabled)
        GPIOIntClear(GPIO_PORTC_BASE,GPIO_PIN_7);      // Clear pending interrupts for PF4
        GPIOIntRegister(GPIO_PORTC_BASE,portExpanderInterruptHandler);     // Register our handler function for port F
        GPIOIntTypeSet(GPIO_PORTC_BASE,GPIO_PIN_7,GPIO_FALLING_EDGE);              // Configure PF4 for falling edge trigger
        GPIOIntEnable(GPIO_PORTC_BASE,GPIO_PIN_7);
        }
        void portExpanderInterruptHandler(){
        if(GPIOIntStatus(GPIO_PORTC_BASE, false)&GPIO_PIN_7){
        if(I2CReceive(0x20,0x0e)&0x01==0x01){
        portExpanderSetOutput(0x01,0xff);
        }
        I2CReceive(0x20,0x10);
        I2CReceive(0x20,0x11);
        GPIOIntClear(GPIO_PORTC_BASE,GPIO_PIN_7);
        }
        }
        void portExpanderpullup(unsigned char port,unsigned char pin){
        I2CSend(0x20,2,(0x0C)+port,pin);
        }

```